Заметки инструктора

Обновлено около 3 лет назад [УДАЛЕНО] в COMP 3321

НЕСЕКРЕТНО //ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

(U) Итак, вы преподаете класс Python. Во что вы ввязались? Вероятно, вам следует потратить несколько минут (или, возможно, несколько дней) на то, чтобы пересмотреть жизненный выбор, который поставил вас в такое положение.

(U) Структура курса

(U) Как упоминалось во введении, этот курс разработан с учетом гибкости. При преподавании в классной комнате один урок или модуль может быть рассмотрен в ходе занятия, которое длится от 45 до 90 минут, в зависимости от затрагиваемых тем. Стандартный способ структурирования курса - это двухнедельный блок с полной занятостью. В течение первой недели десять уроков включают утренние и дневные лекции. В течение второй недели аналогичным образом рассматривается до десяти модулей, по необходимости или просьбе учащихся в классе. (Если потребности класса неизвестны, проголосуйте).
В течение первых нескольких дней занятий учащиеся должны выбрать проект для работы. В последний день студенты должны отчитаться о своих успехах и, по возможности, продемонстрировать свою работу. Преподаватели должны быть доступны вне лекций, чтобы помочь студентам с упражнениями и проектами. (U) Двухнедельный блок - не единственный способ преподавания курса. Материал мог бы быть представлен в более неторопливом темпе, например, во время еженедельного обеда в "коричневом пакете", который продолжается в течение нескольких месяцев.
В качестве альтернативы, если студенты уже подготовлены (или желают провести некоторые из начальных уроков самостоятельно), за двух- или трехдневный семинар можно многого добиться. Например, если все учащиеся уже обладают базовыми знаниями Python, они вполне могут начать с уроков по инструментовке и написанию модулей и пакетов, а затем перейти к рассмотрению различных модулей, представляющих интерес.

(U) Стиль преподавания

(U) При преподавании математики обычная практика, когда преподаватель записывает решения на доске мелом, является сдерживающим методом, который помогает учащимся не отставать. Писать на классной доске не обычно полезно при обучении программированию, но тот же принцип применим; как инструктор, вы должны перенять методы, которые помогут вам замедлиться. Мы рекомендуем, чтобы у вас было живое, интерактивное занятие, размещенное в передней части аудитории, достаточно большое, чтобы его могли видеть все учащиеся. Этот сеанс может быть либо терминальным сеансом, либо записной книжкой Jupyter. Важной деталью является то, что в большинстве случаев команды не должны быть предварительно заполнены; например, вы не должны просто выполнять ячейки из существующего

Записная книжка Джупайтера. Материалы представлены для того, чтобы помочь вам подготовиться, а также в качестве справочного материала для студентов при работе над упражнениями; они не являются приемлемой заменой общему опыту преподавания и усвоения знаний. Вы будете допускать неожиданные ошибки, когда будете писать код вживую перед классом. Не волнуйтесь, расслабьтесь и позвольте студентам помочь вам ~ это поможет им усвоить принципы и понять, как решать свои собственные проблемы. Это не заменяет надлежащую подготовку; слишком много ошибок и промахов приведут к тому, что ваши студенты потеряют интерес к курсу и доверие к вам.

U) Текущая разработка

Разработчики этого курса считают, что текущие материалы разработаны достаточно хорошо, чтобы быть эффективным подспорьем для курса. Однако улучшения, расширения и доработки всегда приветствуются. С этой целью мы попытались упростить внесение вклада в проект.
Документация основана на форке NSAG [УДАЛЕННЫХ] оригинальных материалов COMP 3321. Если вы хотите внести изменения для своих собственных целей, не стесняйтесь клонировать этот репозиторий или разветвлять любой из этих блокнотов в галерее Juptyer. Пожалуйста, отправьте запрос на изменение в Галерее, если вы хотите внести разовый вклад, включая новые или улучшенные упражнения, дополнения к урокам или модулям, или полностью новые уроки или модули. Если вы хотите сотрудничать во всех ноутбуках COMP 3321, свяжитесь с группой COMP3321 GlobalMe и попросите добавить вас.

(U) Возможный ледокол

(U) Чтобы заставить студентов взаимодействовать друг с другом, а также размышлять о коде на абстрактном уровне, рассмотрим следующий ледокол. Инструктор становится человеком-компилятором, интерпретирующим письменные инструкции по выходу из комнаты. (U) Этап I — обсуждение со всем классом (U) Совместно придумайте язык программирования, достаточный для выполнения этой задачи. Четко объясните задачу, показав, с какого места в аудитории инструктор начнет и насколько большим будет шаг. (U) Примите предложения от студентов о том, какими инструкциями они захотят воспользоваться. Запишите каждую инструкцию на доске. Должно быть ясно, что все, что написано на бумаге, должно исходить из синтаксиса на доске. Никакой другой синтаксис не допускается. (U) Убедитесь, что преподаватель и учащиеся точно согласны с тем, что означает каждая инструкция . Как преподаватель, убедитесь, что списка на доске достаточно для решения задачи.
Давайте подсказки, пока это не будет завершено. (U) Пусть учащиеся сами разберутся с синтаксисом. Но вот несколько примеров синтаксиса, которые могут прийти им в голову. - step(n) - принимает целое число и заставляет инструктора выполнять. шаги. - поворот (d) - принимает число в градусах и заставляет инструктора повернуться по часовой стрелке на это много градусов. Учащиеся могут злоупотреблять этим и вводить числа, вызывающие головокружение, например, tum(1440) - препятствие - возвращает логическое значение, указывающее, что препятствие находится прямо перед преподавателем. Также могут быть желательны варианты для проверки влево или вправо. - if < > then < > else < > -- первый пробел принимает логическое значение (убедитесь, что существуют логические функции!). Вторые пробелы содержат инструкции. - в то время как <>:<>-- принимает логическую функцию и любое выражение - не -- выражение для изменения логического значения на любое целое число (U) Этап II -- разбивается на команды (U), как правило, подходят команды из 3-5 студентов. (U) Каждая команда выдает лист бумаги с компьютерными инструкциями для человека-компилятора выйти из комнаты. Допустимый синтаксис - это только тот, который написан на доске. Должно занять не более 15 минут. (U) Этап III - демонстрации (U) По очереди инструктор принимает решение команды

и следует инструкциям, буквально и справедливо. Выходит ли инструктор из комнаты?

(U) Ознакомительное электронное письмо

(U) Есть пара вещей, которые было бы неплохо, если бы студенты могли сделать заранее, в частности, наличие учетных записей GITLAB и внутренних соглашений. У меня есть обучающие программы COMP3321, фасилитаторы которых отправляют зачисленным студентам следующее электронное письмо: (U) Алоха- (U) Вы получаете это электронное письмо, потому что вы зарегистрированы на COMP3321 beginning <..>. (U) Если это вообще возможно, мы могли бы использовать , если вы сделаете несколько подготовительных действий заранее, чтобы в первый день занятия прошли гладко. На самом деле, две простые вещи, которые отнимут у вас минуту времени и сэкономят нам часы в первый день занятий. (U) 1) ПЕРЕЙДИТЕ на iagree и найдите, прочтите пользовательское соглашение INHERE и примите его. (U) 2) ПЕРЕЙДИТЕ В gitlab. (U) Более подробно: (U) 1) Курс будет проходить через LABBENCH и NBGALLERY. Перед использованием вам нужно будет согласиться с условиями предоставления услуг и [УДАЛЕНО] подтвердить это. Инструкции можно найти здесь: https://nbgallery.nsa.ic.gov / (ПЕРЕЙДИТЕ В NBGALLERY), нажав "Получить СКАМЕЙКУ ЗАПАСНЫХ". Нам нужно только, чтобы вы выполнили первый шаг, описанный ниже. Будьте осторожны, продвигаясь дальше, потому что машины LABBENCH самоуничтожаются через две недели после создания, поэтому в первый день занятий вам понадобится новая. (U) СОГЛАШАЙТЕСЬ. Найдите "inhere". Прочитайте пользовательское соглашение INHERE и согласитесь с ним. [УДАЛЕНО] в конечном итоге признаем это соглашение. Это то, что нам нужно. (U) 2) ПЕРЕЙДИТЕ В gitlab. Перейдя туда один раз, будет создана учетная запись. Это все, что нам нужно, прежде чем мы начнем. (U) Если вы хотите пойти дальше и использовать git из командной строки [УДАЛЕНО], следуйте инструкциям здесь: https://wiki.nsa.ic.gov/wiki/Git / Windows (U) Представлено несколько вариантов. Все, что вы можете заставить работать, прекрасно. (U) 3) Необязательно. Класс будет выполняться с LABBENCH. Но некоторые люди предпочитают использовать [УДАЛЕНО] machine. Для этого потребуется выполнить некоторую настройку. Установите Anaconda [УДАЛЕНО], следуя этим инструкциям: https://production.tradecraft.proj.nsa.ic.gov/entry/29475 (U) Если у вас возникли какие-либо проблемы с инструкциями, пожалуйста, свяжитесь с инструктором <..>

(U) Отправка проектов

(U) Возможным способом отправки проектов является GITLAB. [УДАЛЕНО] Преподаватель добавляет всех студентов в классе в группу Gitlab comp3321 [УДАЛЕНО] В рамках этой группы создается новый проект под названием class-projects-Ммм-ГГГГ. Учащиеся отправят свой код в Gitlab. Этого можно легко добиться с помощью веб-приложения, найдя знак "+", скопировав и вставив код. Веб-приложение не позволяет создавать папки. Учащиеся, которым нужны папки, могут создавать их из командной строки или получить помощь в этом.

НЕСЕКРЕТНО //ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

Программирование на Python.

Обновлено 3 месяца назад [УДАЛЕНО] в COMP 3321 (U // FOUO) Введение в курс и учебная программа

для COMP 3321, программирование на Python.

НЕСЕКРЕТНО //ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

(U) История

> ...в декабре 1989,1 искал. проект по программированию "хобби", который будет держать меня занятым в течение недели перед Рождеством. Мой офис ... будет закрыт, но у меня был домашний компьютер, и больше у меня ничего не было. Я решил написать интерпретатор для нового языка сценариев. в последнее время думал о потомке ABC, который понравился бы хакерам Unix / C. Я выбрал Python в качестве рабочего названия для проекта, находясь в слегка непочтительном настроении (и будучи большим поклонником "Летающего цирка" Монти Пайтона) Гвидо ван Россума, Предисловие к " Программированию на Python", 1-е издание.

(U) Мотивация

(U) Python был разработан таким образом, чтобы быть простым и интуитивно понятным без ущерба для мощности, с открытым исходным кодом и подходящим для повседневных задач с быстрыми сроками разработки. Это делает границы между программированием и решением проблем как можно более тонкими. Это подходит для:

> Открытие интерактивного сеанса для решения ежедневных задач,
> Написание сценария, автоматизирующего утомительную и отнимающую много времени задачу,
> создание быстрого веб-сервиса или обширного веб-приложения и
> проведение передовых математических исследований . (U) Если вы еще не знаете ни одного языка программирования , Python - хорошее место для начала. Если вы уже знаете другой язык, то легко освоите Python на стороне. Python не совсем свободен от разочарований и путаницы, но, надеюсь, вы сможете избегать этих частей до тех пор, пока не научитесь хорошо использовать Python. (U) Программирование - это не спорт для зрителей! Чем больше вы будете практиковаться в программировании, тем большему вы научитесь на этом занятии, как вширь, так и вглубь. Python практически обучается сам по себе - цель ваших инструкторов - показать вам все самое интересное и помочь вам двигаться чуть быстрее, чем вы могли бы в противном случае. Счастливого программирования!

U) Цель

(U) Цель этого занятия - помочь студентам легче и надежнее выполнять рабочие задачи, программируя на Python. Чтобы пройти курс, каждый студент должен написать хотя бы одну программу на Python, которая имеет существенную личную полезность или представляет значительный личный интерес. При выборе проекта студентам рекомендуется сначала подумать о задачах, связанных с работой. Для студентов, которым нужна помощь в начале работы, внизу этой страницы приведены несколько предложений по возможным проектам. В первый день инструкторы проведут дискуссию, на которой будут обсуждаться идеи проекта. (U) Этот курс предназначен для студентов с разным образованием и уровнем опыта, от полного новичка до компетентного

программист. Просьба к каждому студенту разработать и реализовать свой собственный проект позволяет каждому учиться и прогрессировать в индивидуальном темпе.

(U) Логистика

(U / / FOUO) Этот курс разработан так, чтобы подходить для самостоятельного изучения. Даже если в вашем расписании нет официальных предложений , вы можете получить доступ к модулям этого курса и работать над ними в любое время.
Даже если у вас нет доступа к. последняя версия Python на рабочей станции или виртуальной машине вы можете получить доступ к персональному блокноту Jupyter, доступному в LABBENCH. Для человека, занимающегося этим вариантом самостоятельного изучения, рекомендуется сначала примерно по порядку изучить основы Python, затем выбрать столько полезных модулей, сколько покажется подходящим. Вы также можете использовать этот блокнот Jupyter для экспериментов и записи решений упражнений. (U / / FOUO) Одна из возможностей для группы потенциальных студентов, начинающих с разным опытом программирования, - использовать Jupyter в качестве инструмента для самостоятельного изучения, пока все не овладеют базовыми знаниями программирования, затем пройти сокращенный курс под руководством инструктора (от двух дней до недели и более, в зависимости от потребностей).

(U) В классе

(U) Для двухнедельного курса: каждый день будет проводиться утренняя лекция и дневная лекция.
Утренняя лекция продлится от часа до девяноста минут; дневная лекция будет несколько короче. Если лекция проходит слишком быстро, пожалуйста, задавайте вопросы, чтобы замедлить нас! Если это происходит слишком медленно, не стесняйтесь работать дальше самостоятельно. (U / / FOUO) Вы будете использовать Python либо в [УДАЛЕННОЙ] среде, либо в LABBENCH. [УДАЛЕНО] Хотя мы укажем на некоторые различия между строками Python 3.x и 2.x, в этом курсе основное внимание будет уделено Python 3. Если вам нужно или вы хотите запустить Python в Linux, возможно, в виртуальной машине MachineShop, мы будем работать с вами. Насколько это возможно, мы будем писать код независимо от платформы и указывать на функции, уникальные для конкретных версий Python. (U) Хотя лекции будут занимать всего два-три часа каждый день, мы призываем вас посвятить остаток дня программированию на Python либо самостоятельно, либо в группах, но по возможности в классе. По крайней мере, один преподаватель будет доступен в классе в обычное рабочее время. (U) Этот курс находится в стадии разработки; мы приветствуем все предложения.
Существует гораздо больше полезных модулей, чем мы можем надеяться охватить за двухнедельный курс; преподаватели проведут голосование, чтобы определить, какие модули охватить. Если есть другая тема, которую вы хотели бы осветить, особенно в духе "Как бы я сделал X, Y или Z в Python?", пожалуйста, спросите - если есть достаточный интерес, мы рассмотрим это в лекции. Мы даже будем рады, если вы внесете свой вклад в документацию курса! Поговорите с преподавателем, чтобы узнать, как это сделать.

(U) Оглавление

(U) Часть II: Полезные модули.Неделя.)

(U // FOUO) Дополнение: Python, HTTPS и LABBENCH

(U) Модуль: Обработка HTML с помощью BeautifulSoup

(U) Модуль: Операции со сжатием и архивами

(U) Модуль: Регулярные выражения

(U) Модуль: Хэши

(U) Модуль: SQL и Python

(U) Дополнение: Простые базы данных с salite3

(U) Модуль: Структурированные данные: CSV. XML и JSON

(U) Модуль: Взаимодействие с системой

(U) Дополнение: Манипулирование. Документы Microsoft Office с модулем win32com

(U): потоковое распространение. и подпроцессы

(U //FOUO). Пакет Python в АНБ

(U) Модуль: введение в машинное обучение (U) Домашнее задание

(U) Домашнее задание дня 1

(U) Домашнее задание дня 2

(U) Упражнения (с решениями)

(U) Упражнения со словарем и файлами

(U) структурированные данные и даты

(U) Упражнения на дату и время

(U) Объектно-ориентированное программирование и исключения

(U) Классные проекты

(U) Нажмите здесь, чтобы перейти к записной книжке, содержащей инструкции по проектам проверки паролей и генератора паролей.

(U) Идеи проекта

(U) Написать скрипт конвертации валюты

(U) Написать веб-приложение

(U) Решить задачи Эйлера проекта

(U) Найти аномальную активность в сети BigCorp

(U) Модуль шифрования RSA, часть 2

(U) Выберите проект из одной из книг Safari ниже

(U) Общие ресурсы

(U) Документация по языку Python

(U) Python 2.7.10

(U) Python 2.7 в DevDocs

(U) Python 3.4.3

(U) Python 3.5 в DevDocs

(U//FOUO) Учебные материалы АНБ

(U) Заметки преподавателя

(U //FOUO) COMP 3321 Сервер изучения Python

(U //FOUO) CRYP 3320 (версия курса для CES)

(U //FOUO) Материалы COMP 3320

(U //FOUO) CADP Python-Ciass

(U) Книги

(U) Автоматизируйте скучные вещи с помощью Python

(U) Black Hat Python: программирование на Python для хакеров и пентестеров

(U) Погружение в Python

(U) Экспертное программирование на Python

(U) С головой в Python

(U) Высокопроизводительный Python (продвинутый)

(U) Изучение Python

(U) Изучение программирования на Python (видео)

(U) Программирование на Python

(U) Ускоренный курс Python (включает 3 примера проектов)

(U) Игровая площадка Python (больше примеров проектов)

(U) Справочник Python Pocket (рассмотрите возможность получения печатной копии)

(U) Программирование на Python для абсолютного новичка (еще больше примеров проектов, игр и викторин)

(U) и больше программ на Python. для абсолютного новичка

(U) Подумайте о книгах по Python

(U) Safari (общий запрос)

(U) Другое

(U) Генератор расписания финального проекта

(U) Просто небольшая записная книжка для случайной генерации расписания для студентов, которые представят свои финальные проекты для COMP3321.

(U // FOUO) Группа Python в NSA GitLab

(U) Руководство автостопщика по Python!

(U //FOUO) python в WikiInfo

(U) Python в StackOverflow (U) Дополнительные целевые ресурсы (часто выдержки из приведенного выше приведены по ссылкам в каждом уроке и модуле.

Урок 01: Введение: Ваша первая программа на Python

Обновлено. несколько месяцев назад [УДАЛЕНО] в COMP 3321 (U) Описывается установка Anaconda, интерпретатор python, базовые типы данных, запущенный код и некоторые встроенные модули.

НЕСЕКРЕТНО //ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

(U) Добро пожаловать в класс!

(U) Давайте узнаем друг друга получше. Встаньте и ждите инструкций. (U) У кого есть конкретный проект на примете?

(U) Способ 1: Установка "Анаконды"

(U) В качестве альтернативы следуйте этой статье tradecraft hub. https://inroduction.tradecraft.proi.nsa.ic.gov / запись/29475 (U) Мы будем использовать версию 4.4.0 дистрибутива Anaconda3 Python, доступную на сайте Get Software. Anaconda включает в себя множество пакетов для крупномасштабной обработки данных, прогнозной аналитики и научных вычислений.

(U) Инструкции по установке

1. (U) Нажмите на ссылку выше
2. (U) Нажмите на кнопку "Загрузить сейчас"
3. (U) Примите соглашение и нажмите "Далее"
4. (U) Нажмите "Далее"
5. (U) Загрузить "Anaconda3-4.4.0-Windows-x86_64.exe"

6. (U) Откройте папку, содержащую загрузку (обычно это ваша папка "Загрузки")
7. (U) Дважды щелкните файл Anaconda3-4-4.0-Windows-x86_64.exe
8. (U) Нажмите "Далее", чтобы запустить программу установки
9. (U) Нажмите кнопку "Я согласен", чтобы принять лицензионное соглашение
10. (U) Выберите установку "Только для меня" и нажмите "Далее"
11. (U // FOUO) Выберите папку назначения на вашем U: диске (например, U:\private\anaconda3)
    i. (U//FOUO) Нажмите кнопку "Обзор ...", нажмите "Компьютер" и выберите U: диск
    ii. (U //FOUO) Выберите "Мои документы" и нажмите "ОК" [Примечание: НЕ создавайте папку с именем anaconda3]
    iii. (U //FOUO) В области ввода папки назначения добавьте "anaconda3" в качестве имени папки
    iv. (U//FOUO) Нажмите "Далее"
12. (U) Нажмите "Установить", чтобы начать установку (оставьте флажки как есть)
13. (U) Подождите около 30 минут, пока установка завершится.

(U) Запуск python

(U) Вы можете запустить python прямо на своем [УДАЛЕННОМ] рабочем столе и сразу же взаимодействовать с ним:

1. (U) Откройте командное окно Windows (введите "cmd" в строке поиска программ Windows)
2. (U) Введите "python" в окне командной строки

(U) Запуск Jupyter

(U) В качестве альтернативы вы можете запустить python в браузере из python с поддержкой Интернета, который называется jupyter записная книжка. [УДАЛЕНО] Галерея записных книжек находится на сайте go nbgallery. Однако для этого класса каждый из нас запустит свой собственный веб-портал Jupyter для управления записными книжками наших классов.

1. (U) В меню "Пуск" Windows найдите "jupyter"
2. (U) Щелкните правой кнопкой мыши на Jupyter Notebook в результатах и выберите "Свойства"
3. (U //FOUO) В поле "Цель" добавьте " U:\private" в конце (после "записная книжка") [Примечание: не забудьте пробел перед U:\private]
4. (U) Нажмите Применить, а затем ОК
5. (U) Снова найдите jupyter в меню "Пуск" и нажмите на Jupyter Notebook, чтобы запустить его
6. (U) Подождите несколько мгновений… После этого Jupyter должен запуститься в вашем браузере по адресу http://localhost:8888 / дерево

(U) Метод 2: настройка лабораторного стола

(U//FOUO) Шаг 1: Доступ к LABBENCH

(U // FOUO)]перейти я согласен, прочитал и принимаю Настоящее Пользовательское соглашение. (U //FOUO) Это

необходимое условие для доступа к LABBENCH, виртуальной системе, в которой мы будем работать. Может потребоваться несколько часов, чтобы одобрение распространилось по системе.

(U) Шаг 2: Посетите галерею Jupyter

1. (U//FOUO) иди быстрее
2. (U) Нажмите на логотип галереи Jupyter, чтобы перейти в галерею.
3. (U) Нажмите "Совершить экскурсию по галерее" для быстрой демонстрации.
4. (U / / FOUO) Чтобы найти тетради с курсами, выполните поиск по "Программе" или выберите "Тетради" > Обучение > COMP 3321 и отсортируйте по названию, чтобы найти учебную программу.

(U//FOUO) Шаг 3: Настройте Jupyter на LABBENCH

(U // FOUO) В галерее Jupyter нажмите "Jupyter на LABBENCH", чтобы получить руководство по настройке.

(U) Базовые основы: данные и операции

(U) Наиболее базовыми типами данных в Python являются:

Числа

Целое число <type 'int'> (это "произвольная точность"; не нужно беспокоиться, 32 бита это

или 64 бита и т.д.)

Float <type'float'>

Complex <type' complex'> (используя 1j для мнимого числа)

Строки <тип 'str'>

Нет разницы между одинарными и двойными кавычками

Экранирование специальных символов (например, кавычек)

Необработанная строка r'raw string' предотвращает необходимость некоторых экранирований

Тройные кавычки позволяют использовать строки из нескольких строк

Unicode u'Albert \x26 Ernie' <type' unicode'>

Логические значения: True и False (U) Мы оперируем данными, используя

операторы, например математические операторы +, - ; ключевое слово in и другие

функции, которые представляют собой операции, которые принимают один или несколько фрагментов данных в качестве аргументов, например,

type ('hello'), len ('world') и

методы, которые присоединяются к фрагменту данных и вызываются из него с помощью а . чтобы отделить

данные от метода, например, 'Hello World'.split() или 'abc'.upper()

(U) Глубоко в недрах Python все это, по сути, одно и то же, но синтаксически и

педагогически имеет смысл разделить их.

(U) Фрагменты базовых данных могут храниться внутри контейнеров, в том числе

Спискинаборов

словарей

но мы представим их позже.

(U) Интерактивный переводчик

Имея этот базовый опыт, давайте попробуем кое-что сделать в вашем командном окне Windows ...

```
U:\private > python
Python 3.5.1 |Anaconda 2.5.0(64-разрядная версия)| (по умолчанию, январь 29 2016, 15:01:46) [ MSC v.1900 64-разрядная версия (AMD64)] в win32
Введите "справка" , "авторское право" , "кредиты" или "лицензия" для дополнительная информация.
5 + 7
тип (5+7)
4.5 - 5.5
тип (7.1 - 2.1)
# это изменилось в python3
13/5
13//5
1j * 4j
"привет" + " мир"
"привет " * 10
```

(U) Выполнение кода в файле

Откройте файл first-program.py (или что-нибудь, заканчивающееся на .py) в вашем любимом редакторе (я использую emacs,

но вы можете использовать все, что захотите). (U) Если у вас нет любимого редактора, сделайте это:

1. (U) Перейдите на свой портал Jupyter по адресу http://localhost:8888/tree
2. (U) Откройте меню кнопки "Создать" и выберите "Текстовый файл"
3. (U) Щелкните по имени "Untitled.txt" и введите новое имя файла как "first-program.py" (U) Введите в нем несколько инструкций Python:

```
5+7
9*43
8**12
```

(U) Не забудьте сохранить это (Файл-> Сохранить из Jupyter). (U) Чтобы запустить его, укажите имя файла в качестве

аргумента Python: убедитесь, что окно командной строки ссылается на ту же папку, что и файл.

То есть.:\private для большинства. Если ваше командное окно не ссылается на U:\private, сделайте это:

1. Введите "U:"
2. Введите "cd private"

```
U:\private > python first-program.py
```

(U) Похоже, ничего не происходит, потому что выполняется автоматическая печать выходных данных. функция выполняется только в интерпретаторе. Исправьте это:

```
печать(5+7)
печать(9*43)
печать(8**12)
```

(U) Встроенные функции и методы

(U) Некоторые функции работают практически с любыми аргументами, которые вы предоставляете:

справка(x): показывает интерактивную справку

каталог (x): предоставляет каталог объекта, т.е. все доступные методы

тип (x) : указывает тип x — тип почти такой же, как у любого другого объекта

isinstance(a,b) : указывает, является ли объект a экземпляром b, который должен быть типом; что-то вроде type(a) == b

print

hasattr(a,b) : указывает, есть ли у a что-то с именем b; что-то вроде b в dir(a)

getattr

id

ввод

(U) Функции конструктора обычно стараются сделать все возможное с приведенными вами аргументами и возвращают соответствующие данные запрошенного типа:

str : преобразует числа (и другие вещи) в их строковые представления

int : усекает значение с плавающей точкой, анализирует строки, содержащие одно целое число, с необязательным основанием (т.Е. base), ошибка при комплексном представлении

float: анализирует строки, выдает float-представление int, ошибка при сложном

сложный : принимает (реальные, воображаемые) числовые аргументы или анализирует str для одного числа (U) Другие функции работают только с одним или двумя типами данных:

Цифры:

Функции: abs, round, float, max, min, pow (модульный), chr, divmod и т.д.

Операторы: стандартные математические, побитовые: <<,>>,&, |, ^, ~

Методы: Числовые классы не имеют методов

Строки:

Функции: len, min, max, ord

Операторы: +, * (с числом), в

методах: strip, split, startswith, upper, find, index и многих других; используйте dir ('любая строка'), чтобы найти

больше

(U) Упражнения:

1. Составьте. список покупок из пяти вещей, которые вам понадобятся в продуктовом магазине. Поместите каждый элемент в отдельную строку в ячейку. Не забудьте использовать кавычки! Используйте функцию print(), чтобы отображался каждый из ваших элементов (попробуйте сначала без нее).
2. Время доставки продуктов указано в 9.42, 5.67, 3.25, 13.40 и 7.50 соответственно. Используйте Python как удобный калькулятор, чтобы сложить эти суммы.
3. Но подождите! Вы решаете купить пять из последнего товара. Повторно рассчитайте ваше общее количество.
4. Используя функцию len(), определите количество символов в строке "насыщение крови кислородом" функциональная магнитно-резонансная томография в зависимости от уровня" (интересный факт: эта строка является самой длинной записью в индексе WordNet 3.1).
5. Выберите свою любимую закуску. С помощью оператора * распечатайте ее в количестве 100 экземпляров. Измените свой код, чтобы они печатались с пробелами между ними.
6. Задача: Запустите dir('любая строка'). Выберите два метода, которые кажутся интересными, и запустите справку ('any string' .interesting_method) для обоих из них. Можете ли вы понять, как использовать эти методы?
7. Бонусная задача: Можете ли вы выяснить, как получить тот же результат, что и при выполнении упражнения? используя только один оператор печати? Если да, можете ли вы также выполнить это в одной строке кода?

НЕСЕКРЕТНО //ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

Урок 02: Переменные и функции

Обновлено 5 месяцев назад [УДАЛЕНО] (U) Введение в переменные и функции в Python.

НЕКЛАССИФИЦИРОВАНО

(U) Мое Королевство для переменной

(U) Все числа и строки в мире не принесут вам никакой пользы, если вы не сможете уследить за ними. Переменная в Python - это имя, которое привязано к чему-либо - числу, строке или чему-то более сложному, например списку, объекту или даже типу. (U) Python имеет динамическую типизацию, что означает, что переменная, однажды объявленная, может содержать различные типы данных в течение всего срока службы. Переменная объявляется с помощью оператора * . Давайте, дайте этому значению имя!

```
x = 2
x + 5
```

(U) В интерактивном режиме a _ - это ссылка на последний вывод ячейки.

```
9*8
```

(U) Это может быть особенно полезно, когда вы забываете сохранить результат длинного вычисления, присвоив ему имя.

```
y = x + 5
```

(U) Обратите внимание, что строка y = x + 5 не произвела вывода, поэтому была проигнорирована при вызове _.

```
isinstance(x, int)
isinstance(x, str)
```

(U) Итак, давайте изменим, чему равен x (и даже изменим его тип!), Просто переназначив переменной name.

```
x =      "Привет"
isinstance(x, int)
isinstance(x, тип(8))
isinstance(x, тип(                    'a'     ))
y = x + 5
```

(U) Так что насчет преобразования из одного типа в другой?

```
a =      "3.1234"
тип(a)
b = с плавающей точкой(a)
тип(b)
b
                    # Это должно провалиться. Почему?
с плавающей точкой(x)
c = str(b)
c
```

```
i = int(b)
i
```

(U) Продолжайте и используйте функцию dir (), чтобы увидеть, какие переменные вы определили. Эта команда показывает все объекты, которые определены в вашей текущей области видимости (мы поговорим о области видимости позже).

```
dir()
del x
dir()
x + 5 # Почему это происходит сбой?
```

(U) Мы также можем назначать переменные с помощью некоторых причудливых сокращений:

```
a = b = c = 0
печать(a)
печать(b)
печать(c)
x, y = l, 2
печать(x)
печать(y)
z = 1,2 # Что это дает??
z
x, y, z = 1, 2 # Как насчет этого?
```

Обратите внимание, что с момента сбоя последней команды значения x, y и z не изменились.

```
вывести(x)
вывести(y)
x, y = y, x # Быстрая замена!
печать(x)
печать(y)
```

(U) Имена переменных могут быть присвоены всем различным типам объектов. Помните об этих хитростях, когда будете знакомиться с более сложными типами. (U) Давайте на минутку обсудим списки. Мы подробнее расскажем о контейнерах позже в курсе, но вам нужно знать основы для выполнения одного из упражнений.

```
l = [1, 2, 3, 4]
1 в l
5 в l
l = [ "один" , "два" , "три" , "четыре" ]
"один" в l
```

(U) Упражнения

1. Сохраните копию вашей любимой закуски в переменной. Используя эту переменную, распечатайте свой перекус 100 раз.

2. Спросите своего соседа, какая у него любимая закуска. Сохраните это в. переменная. Теперь у вас должно быть две переменные, содержащие закуски. Сложите (конкатенируйте) их вместе и выведите результат 100 раз.

3. Используя приведенное выше обозначение [ ], составьте список из пяти продуктов и сохраните в переменной. (Если вы ранее выполняли упражнение со списком продуктов, используйте эти продукты). Используя переменную из Exercise., проверьте, есть ли ваша любимая закуска " в" списке.

4. Используя список покупок из упражнения 3 и переменную из упражнения 2, проверьте, есть ли в вашем списке любимая закуска вашего соседа так же, как и в случае с вашим перекусом.

5. Используйте функцию "быстрая замена", чтобы поменять свою любимую закуску на закуску вашего соседа. Выведите обе переменные, чтобы увидеть результат. Вы довольны или огорчены своей новой любимой закуской?

(U) Функции

(U) Итак, что еще мы можем сделать с переменными? Много!

```
7 % 2        # Оператор по модулю
7 ** 2
мин(2,7)             # встроенная функция
макс(2,7)
реж.( "a" )
```

(U) Python поставляется с кучей встроенных функций. Мы уже использовали некоторые из них: dir(), min(), max(), isinstance() и type(). Python включает в себя гораздо больше, например:

```
abs(-1)
round(1.2)
len( "12345" )
```

(U) Но функции занимают память, а в Python включены сотни модулей, поэтому мы не можем иметь доступ ко всему, что Python может делать сразу. Чтобы использовать функции, которые не встроены, мы должны сказать Python, чтобы он их загрузил. Мы делаем это с помощью инструкции import:

```
import os
```

(U) При этом загружается модуль операционной системы. Модуль представляет собой файл, содержащий определения и инструкции, и обычно используется для хранения набора связанных функций. Модуль os содержит функции, относящиеся к операционной системе компьютера, на котором запущен Python. (U) Итак, что же содержится в os? Давайте посмотрим:

```
dir(os)
```

(U) Это дает вам список всего, что определено в модуле OS,

```
os.name          # почему имя не требует круглых скобок?
os.listdir()
```

(U) Python имеет надежную документацию по стандартным модулям. Всегда консультируйтесь с документацией, если вы не уверены, как использовать функцию. (U) Что, если мне не нужно все в модуле?

```
из      ос    импорт      listdir
listdir()
```

(U) Мы перейдем к другим модулям позже на занятии. Сейчас мы просто коснемся двух других: sys, который содержит переменные и функции, относящиеся к взаимодействию Python с системой; и random, который обеспечивает генерацию случайных чисел.

```
импорт      sys
каталог (sys)
sys.argv            # содержит аргументы командной строки
sys.exit()          # завершает работу с Python.возможно, вы не захотите вводить это)
импорт      случайный
random.randint(l, 5)
случайный.random()
```

(U) Упражнения

1. Составьте список цен на продукты (9,42, 5,67, 3,25, 13,40 и 7,50 соответственно) и храните в переменной . Используйте встроенные функции, чтобы узнать цену на самый дешевый и самый дорогой товар из вашего списка покупок.
2. импортируйте random и запустите справку (random.randint). Используйте randint для случайной печати от 0 до 100 экземпляров вашей любимой закуски.
3. Запустите dir (случайным образом). Найдите функцию в random, которую вы можете использовать для возврата случайного товара из вашего списка покупок. Помните, что вы можете воспользоваться функцией help(), чтобы узнать, что делают различные функции!
4. Введите код для случайного выбора цены из вашего списка цен на продукты, округлите до ближайшего целое число и выведите результат.
5. Задача: в вашем продуктовом магазине проводится странная акция под названием "выиграй бесплатную сдачу"! Выбирается случайный товар из вашего (прайс-листа), и вы платите 10 долларов. Если товар стоит меньше 10 долларов, вы получаете свой товар и сдачу обратно в обычном режиме; однако, если вам повезет и цена будет больше 10 долларов, вы получаете товар и разницу в цене обратно в виде сдачи. Написать код

случайным образом выберите цену из своего прайс-листа и распечатайте сумму сдачи, которую кассир должен заплатить вам в ходе этой акции. Подсказка: воспользуйтесь встроенной функцией abs.

(U) Создание собственных функций

(U) Функции (в Python) на самом деле являются просто специальными переменными (или типами данных), которые могут иметь входные данные и выходные данные. После определения вы можете обращаться с ними как с любыми другими переменными. (U) Функции определяются с помощью определенного синтаксиса:

Начните с ключевого слова def,

за которым следует имя функции и

список аргументов, заключенных в (), затем

строка заканчивается символом :, и

тело функции в следующих строках имеет отступ.

(U) Python использует пробелы для определения блоков, в отличие от., Java и других языков, которые используют {} для этой цели. (U) Для получения выходных данных из функции используется ключевое слово return, за которым следует то, что должно быть возвращено. Если вывода нет, используйте return сам по себе или просто не используйте его.

```
определение first_func (x) :
    возврат x*2

first_func(10)

first_func( "привет" )
```

(U) Вау!… Python ДЕЙСТВИТЕЛЬНО не заботится о типах. Вот самая простая функция, которую вы можете написать на Python (без ввода, без вывода и ничего больше!):

```
определение простой ():
    пас # или вернуть
simple() # СКУЧНО!
```

(U) Давайте немного поиграем с новой функцией… мы будем называть эту мощную функцию add .

```
определение добавить(a,b):
    вернуть a+b
добавить(2, 3)
добавить(1)
добавить( 'a' ,3)
добавить( , 'b' )
добавить
def add2 (a, b):
```

```
        вывести (a+b)

x = добавить(2, 3)          # Что это дало?
x

x = add2(2, 3)              # Что это дало?
x
```

(U) Не забывайте: имена функций тоже являются переменными.

```
x = добавить    # Что это дало?
добавить = 7    # А это?
добавить(2,3)     # Мы сломали эту функцию. Урок здесь.
x(2,3)
```

(U) Упражнения

1. Напишите функцию all_the_snacks, которая берет snack (строку) и использует оператор * для ее печати
   100 раз. Протестируйте свою функцию, используя каждый из пунктов в вашем списке покупок. Что произойдет, если вы введете. число в свою функцию? Результат соответствует вашим ожиданиям?
2. Возможно, вы заметили, что ваша функция all_the_snacks печатает все ваши закуски в расплющенном виде
   вместе. Перепишите all_the_snacks так, чтобы он принимал дополнительный разделитель аргументов . Используйте + объедините свой параметр snack и разделитель перед умножением. Протестируйте свою функцию с разными входными данными. Что произойдет, если вы используете строки как для закуски, так и для разделителя? Оба числа? Строка и целое число? Это то, чего вы ожидали?
3. Перепишите all_the_snacks так, чтобы он также принимал переменную num, которая позволяет вам настраивать количество раз, когда ваша закуска распечатывается.
4. Напишите функцию in_grocery_list, которая принимает в grocery_item значение True или False в зависимости от от того, есть ли товар в вашем списке.
5. Напишите функцию price_matcher, которая не принимает аргументов, но печатает. случайный продуктовый товар и. случайная цена из вашего прайс-листа при каждом запуске.
6. Задача: измените свой price_matcher, чтобы возвращать товар с ценой, а не распечатывать их. Напишите функцию free_change, которая вызывает ваш новый pricejnatcher и использует результат для печати вашего товара и абсолютного значения изменения для товара, предполагающего, что вы заплатили.10.

(U) Аргументы, ключевые аргументы и значения по умолчанию

```
def  f (a, b, c):
        возврат  (a + b) * c
```

(U) Вы можете присвоить аргументам значения по умолчанию. Это делает их необязательными.

```
определение f(a, b, c=1):
        возврат (a + b) * c
f(2, 3)
f(2, 3, 2)
```

(U) Вы также можете называть аргументы по имени,

```
f(b=2, a=5)
f(b=2, c=5)
определение g(a=1, b, c):
        возврат (a + b) * c                # Что здесь происходит?
```

(U) Упражнения

1. Перепишите all_the_snacks так, чтобы num и spacer по умолчанию имели значения 100 и ', ' соответственно.

   Используя вашу любимую закуску в качестве входных данных, попробуйте запустить вашу функцию без дополнительных входных данных.

2. Попробуйте запустить all_the_snacks с вашей любимой закуской и разделителем "! " и без дополнительных

   входных данных. Как бы вы запустили его, вводя вашу любимую закуску и 42 для num , сохраняя

   значение по умолчанию для spacer ? Можете ли вы использовать этот метод для ввода пробела и числа в обратном порядке?

(U) Область применения

(U) В программировании область применения является важной концепцией. Это также очень полезно, позволяя нам иметь

гибкость при повторном использовании имен переменных в определениях функций.

```
x = 5
def f ():
        x = 6
        выведите(x)
x
f()
x
```

(U) Давайте поговорим о том, что здесь произошло. Всякий раз, когда мы пытаемся получить или изменить значение. variable ,

Python всегда ищет эту переменную в наиболее подходящей области видимости.closest).

(U) Итак, в функции выше, когда мы объявляли. = 6, мы объявляли. локальная переменная в

определении. . Это не изменило global. за пределами функции. Если вы хотите, чтобы это

произошло, просто используйте ключевое слово global.

```
x = 5
def f ():
```

```
        глобальный  x
        x = 6
x
f()
x
```

(U) Будьте осторожны с областью видимости, она может позволить вам делать некоторые вещи, которые вы, возможно, не захотите. или, возможно, вы делаете!), например, переопределять встроенные функции.

```
len(    "моя строка длиннее 3"                  )
def     len    (x):
        Возврат       3
len(    "моя струна длиннее 3"                  )
```

(U) ввод

(U) Функция ввода - это быстрый способ получить доступ к данным из stdin (пользовательский ввод). Он принимает необязательный строковый аргумент, который является приглашением, выдаваемым пользователю, и всегда возвращает строку. Достаточно просто!

```
a = входные данные (  "Пожалуйста, введите свое имя: "          )
a
```

(U) расширенные аргументы функции

(U) В большинстве случаев вы знаете, что хотите передать в свою функцию. Иногда бывает полезно принимать произвольные аргументы. Python позволяет вам сделать это, но для этого требуется немного синтаксического сахара, который мы раньше не использовали.

- Распаковка списка и словаря
- Упаковка списка и словаря в аргументы функции

(U) Упражнения

1. С помощью ввода задайте свой любимый цвет и сохраните его в переменной my_color. Используйте входные данные, чтобы запросить любимый цвет вашего соседа и сохраните его в переменной neighbour_color.
2. Используйте input, чтобы запросить ваш любимый номер и сохранить его в переменной my_num. Запустите 2 + my_num . Почему это не удается? Как вы можете это исправить?
3. Пишите. Функция color_swapper "Первоапрельского розыгрыша", которая принимает my_color и neighbour_color в качестве входных данных и печатает. сообщение с указанием ваших любимых цветов и цветов вашего соседа соответственно. Добавьте строку перед печатью, которая меняет местами содержимое переменных, так что теперь

сообщение печатается с заменой ваших любимых цветов. Запустите свою функцию, а затем распечатайте содержимое my_color и neighbour_color. Как вы смогли поменять их местами в функции, не меняя местами в своем блокноте?

4. Задача: напишите global_color_swapper, который меняет ваши цвета местами по всему миру. Запустите вашу функцию и затем распечатайте содержимое my_color и neighbour_color . Почему это может быть плохой идеей, даже для первоапрельской шутки?

(U) Обзор

1. Напишите функцию с именем 'Volume', которая вычисляет и возвращает объем коробки с учетом ширины, длины и высоты.
2. Напишите функцию с именем 'Volume2', которая вычисляет объем коробки, предполагая, что высота равна 1, если не задано.
3. Сложная задача: импортируйте модуль 'datetime'. Поэкспериментируйте с различными методами. В частности, определите, как выводить текущее время.

Урок. Функциональные упражнения

Создано почти 3 года назад [УДАЛЕНО] в COMP 3321 (U) Функциональные упражнения для COMP3321

Урок.

(U) Урок 2 - Функциональные упражнения

(U) Напишите функцию isDivisibleBy7(num), чтобы проверить, равномерно ли число делится на 7.

```
>>>  isdivisible7(21)
Верно
>>>  isdivisible7(25)
Ложь
```

(U) Напишите функцию isDivisibleBy(число, делитель), чтобы проверить, равномерно ли число делится на делитель.

```
>>>  Разделяется на две части(35,7)
Верно
>>>  раЗделяемосТь(35,4)
Ложь
```

(U) Создайте функцию shout (word), которая принимает строку и возвращает эту строку заглавными буквами с восклицательным знаком.

```
>>> крик( "бананы" )
"БАНАНЫ!"
```

(U) Создайте функцию introduct(), которая запрашивает у пользователя его имя и выкрикивает его в ответ. Вызовите свою функцию shout, чтобы это произошло.

```
>>> Как тебя зовут?
>>> Боб
ПРИВЕТ, БОБ!
```

Урок 2. Вариативные упражнения

Создано почти 3 года назад [УДАЛЕНО] в COMP 3321 (U) Переменные упражнения для COMP3321 урок.

(U) Урок 2. Упражнения с переменными величинами

(U) Определите тип каждой из следующих переменных и добавьте тип после каждой переменной в. комментарий.

```
a = 2999
b = 90,0
c = "145"
d = "\u0CA0_\u0CA0"
e = "Верно"
f = Верно
g = len( "образец" )
h = 100 ** 30
i = 1 >= 1
j = 30%7
k = 30/7
l = b + 7
m = 128 << 1
n = bin(255)
o = [m,l, k, n]
p = len(o)
```

(U) Какое значение имеет переменная my_var в конце этих назначений? Добавить. сравнение после последнего утверждения в виде my_val ==

```
my_var = 99
my_var += 11
my_var = str(my_var)
```

```
my_var *= 2
my_var = len(my_var)
my_var *= 4
```

Урок 03: Управление потоком

Обновлено более 1 года назад [УДАЛЕНО] в COMP 3321 (U) Управление потоком Python с условными выражениями и циклами.if, while, for, range и т.д.).

НЕКЛАССИФИЦИРОВАНО

(U) Введение

(U) Если вы когда-либо программировали раньше, то знаете, что одним из основных строительных блоков алгоритмов является управление потоком. Он сообщает вашей программе, что делать дальше, в зависимости от состояния, в котором она находится в данный момент.

(U) Сравнения

(U) Сначала давайте посмотрим, как сравнивать значения. Операторами сравнения являются >, >=, <, <=, ! =, и == . При работе с числами они делают то, что вы думаете: возвращают True или False в зависимости от того, является ли утверждение true или false.

```
2 < 3
2 > 5
x = 5
x == 6
x != 6
```

(U) Python 2.x позволит вам попробовать сравнить любые два объекта, независимо от того, насколько они отличаются. Результаты могут оказаться не такими, как вы ожидаете. Python 3.x сравнивает типы только там, где. определена операция сравнения.

```
"яблоко"        >    "оранжевый"      # в алфавитном порядке с учетом регистра
"яблоко"        >    "Оранжевый"
"яблоко"        > [     "оранжевый"   ]
"яблоко"        > (     'оранжевый'    ,)
```

(U) Мы оставим дальнейшее обсуждение сравнений на потом, в том числе о том, как разумно сравнивать создаваемые вами объекты.

(U) Упражнения

1. Напишите функцию you_won, которая случайным образом выбирает число из вашего прайс-листа (9.42, 5.67, 3.25, 13.40 и 7.50) и выводит значение True или False в зависимости от того, больше ли случайное число 10.
2. Напишите функцию snack_check, которая принимает строку snack и возвращает True или False в зависимости от того, является ли это вашей любимой закуской.

(U) Условное выполнение: оператор if

(U) Оператор if - важный и полезный инструмент. В основном он гласит: "Если условие истинно, выполните запрошенные операции".

```
def четный(n):
    если(n % 2 == 0):
        напечатать("Я квитанция!" )
четный(2)
четный(3)
четный( "привет" ) # Это было глупо
```

(U) Что, если мы захотим иметь возможность сказать, что мы не квиты? Или отправленный пользователем. неверный шрифт? Мы используем предложения else и elif.

```
def четный(n):
    если(введите(n) != int):
        печать( "Я говорю только о целых числах" )
    elif (n % 2 == 0):
        напечатать("Я квитанция!" )
    остальное:
        напечатать("Я странный!" )
четный(2)
четный(3)
четный( "привет" )
```

(U) Упражнения

1. Перепишите snack_check, чтобы использовать строку snack, и выведет соответствующий ответ в зависимости от того, является ли вводимая информация вашей любимой закуской или нет.
2. Напишите функцию in_grocery_list, которая принимает распечатки grocery_item. другое сообщение в зависимости от того, есть ли grocery_item в вашем списке покупок.
3. Измените in_grocery_list, чтобы проверить, является ли grocery_item строкой. Выведите сообщение, предупреждающее пользователя, если это не так.

4. Задача: перепишите функцию you_won так, чтобы она выбиралась случайным образом. выберите номер из вашего прайс-листа и распечатайте соответствующее сообщение в зависимости от того, выиграли вы или нет.число было больше 10) или нет. Также укажите сумму сдачи, которую вы получите в своем сообщении. (Напомним, что вы выигрываете сумму сдачи, которую вы должны были бы получить ...).
5. Сложная задача: Написать. функция, которая импортирует дату и время и использует их для определения текущего времени. Эта функция должна напечатать соответствующее сообщение в зависимости от времени, например: если текущее время находится между 09.00 и 1000, выведите сообщение "Время утренней лекции!"

(U) Циклическое поведение

(U) Цикл while

(U) Цикл while используется для повторяющихся операций, которые продолжаются до тех пор, пока выражение истинно. (U) Знаменитый бесконечный цикл:

```
пока     (2 + 2 == 4):
        печать(     "навсегда"        )
```

(U) Ошибка, которая может привести к бесконечному циклу:

```
i = 0
пока     (i <= 20):
        вывести(i)
```

(U) Вероятно, более разумно ввести следующее.

```
i = 0
пока     (i <= 20):
        вывести(i)
        i += 1
```

(U) прерваться и продолжить

(U) Для большего контроля мы можем использовать break и continue (они работают так же, как в C). Команда break завершит наименьший цикл while или for:

```
i = 0
пока ( Верно ) :
    i += 1
    вывести(i)
    если(i == 20):
        перерыв
```

(U) Команда continue остановит текущую итерацию цикла и перейдет к следующему значению.

```
i = 0
пока ( Верно ) :
    i += 1
    если(i == 10):
        вывести( "Мне 10!" )
        продолжить
    вывести(i)
    если(i == 20):
        прервать
```

(U) Предложение else

(U) У вас также может быть оператор else в конце цикла. Он будет выполняться, только если цикл завершится нормально, то есть когда результатом условного выражения будет False . Перерыв позволит пропустить это.

```
i = 0
пока (i < 2):
    вывести (i)
    i += 1
остальное:
    напечатать('Это выполняется после того, как условие становится ложным'. )
печать( "Готово!" )

i = 0
while (i < 2):
    выведите (i)
    if True :
        перерыв
    i += 1
    остальное:
        напечатать('Это не будет напечатано, потому что цикл был завершен раньше'. )
распечатать('Готово!" )
```

(U) Упражнения

Подсказка: для выполнения этих упражнений вам не потребуется продолжения или перерыва.

1. Ранее мы распечатали множество копий. string с помощью оператора /* . Используйте цикл. while, чтобы распечатать 10 копий вашей любимой закуски. Каждая копия может быть на своей отдельной строке, это нормально.
2. Смешивайте и сочетайте! Напишите цикл while, который использует /* для печати нескольких копий ваших любимых закусок в строке. Распечатайте 10 строк с количеством копий на строку, соответствующим номеру строки .в вашей первой строке будет одна копия, а в последней - 10).
3. Задача: Напишите цикл while, который печатает 100 копий вашей любимой закуски в одной единственной (обернутой) строке. Подсказка: используйте + .

(U) Цикл for

(U) Цикл for, вероятно, является наиболее используемым элементом потока управления, поскольку он обладает наибольшей функциональностью. В нем в основном говорится: "для следующих явных элементов сделайте что-нибудь". Мы собираемся использовать тип списка здесь. Более интересные свойства этого типа будут описаны в другом уроке.

```
для i в [1,2,3,4,5, 'a' , 'b' , 'c' ]:
    выведите(i)
```

(U) переменную. "становится" каждым значением списка, после чего выполняется следующий код:

```
для i в [1,2,3,4,5, 'a' , 'b' , 'c' ]:
    печать (i, type(i))
для c в "оранжевом" :
    вывести(c)
```

(U) Упражнения

1. Напишите цикл for, который выводит каждый символ в строке "уровень оксигенации крови" зависимая функциональная магнитно-резонансная томография" (интересный факт: эта строка является самой длинной записью в индексе WordNet3.1).
2. Составьте список покупок из пяти пунктов.или создайте один). Напишите. цикл for для распечатки сообщения.Заметка для себя "купите":"а затем продуктовый товар.
3. Напишите цикл for, который будет распечатан. нумерованный список ваших продуктовых товаров.
4. Очевидно, что ваша любимая закуска важнее других блюд из вашего списка. Измените значение для цикл из упражнения 3, чтобы использовать печать с остановкой после того, как вы нашли свою любимую закуску в своем списке. Вопрос: Могли бы вы достичь того же результата, не используя перерыв? Бонус: если вашей закуски нет в списке, попросите ваш код напечатать предупреждение в конце.
5. Задача: используйте метод разделения строк, чтобы написать цикл for, который выводит каждое слово в строка "функциональная магнитно-резонансная томография, зависящая от уровня оксигенации крови". Подсказка: запустите справку(str.split)

(U) для заполнения цикла: диапазон и xrange

(U) Ок, это здорово ... но я хочу напечатать 1 000 000 чисел! Функция range возвращает список значений на основе предоставленных вами аргументов. Это. простой способ сгенерировать значения от 0 до 9:

```
печать (диапазон(10))

для     i    в      диапазоне (10):
        печать (i)
для     i    в      диапазоне (10, 20):
        печать (i)
для     i    в      диапазоне (10, 20, 2):
        печать (i)
для     i    в      диапазоне (100, 0, -5):
        распечатайте (i)
```

(U) Получается. отличный инструмент для хранения. понятие индекса цикла!

```
a =     "mystring"
для     i    в      диапазоне (len(a)):
        печать(     "Символ в позиции "                    + str(i) +          "есть"          + a[i])
```

(U) Кстати, функция enumerate является предпочтительным способом отслеживания индекса цикла:

```
для   (i, j)          в      перечислить(a) :
        вывести(    "Персонаж на позиции "                 + str(i) +          "есть"          + j)
```

(U) В Python функция range создает итератор. На данный момент представьте итератор как объект, который знает, с чего начать, где остановиться и как перейти от начала к остановке, но не отслеживает каждый шаг на этом пути сразу. Позже мы подробнее обсудим итераторы. (U) В Python., xrange действует как Python.'s range . range в Python. производит. list , поэтому весь диапазон выделяется в памяти. Вы почти всегда должны использовать xrange вместо range в Python ..

```
b = диапазон (100000000)              # О, это было быстро
б    # Это всего лишь объект!
для     i    в      диапазоне (10000):
        если(i % 2 == 0):
                вывести(i)

b = диапазон (0, 1000000, 100)

b

b[0]
```

```
b[1]

b[2]

b[-1]
```

(U) Упражнения

1. Используйте range для написания цикла for для печати нумерованного списка покупок.
2. Используйте enumerate, чтобы распечатать пронумерованный список покупок. Теперь вы сделали это тремя способами. В чем плюсы и минусы каждого метода? Часто существует несколько разных способов получить одинаковый результат! Однако обычно один из них более элегантен, чем другие.
3. Используйте range, чтобы написать цикл for, который распечатает 10 копий вашей любимой закуски. Как это сравнивается с использованием цикла while?
4. Задача: Напишите игру "Угадай мой номер", которая генерирует случайное число и дает вашему пользователю фиксированное количество угадываний. Используйте входные данные, чтобы узнать предположения пользователя. Подумайте о том, какой тип цикла вы могли бы использовать и как вы могли бы предоставить обратную связь на основе предположений пользователя. Подсказка: какой тип возвращается при вводе? Возможно, вам потребуется преобразовать это значение в более удобный тип… Однако, теперь, что произойдет, если ваш пользователь введет что-то, что не является числом?

НЕКЛАССИФИЦИРОВАНО

Урок 3. Упражнения по управлению потоком

Обновлено почти 3 года назад [УДАЛЕНО] в COMP 3321 (U) Упражнения по управлению потоком для COMP3321

Урок.

(U) Урок 3. - Упражнения по управлению потоком

(U) Измените приведенный ниже цикл так, чтобы он выводил числа от 1 до 10.

```
для i в диапазоне (9):
    печать (i)
```

(U) Используя цикл for и enumerate, напишите функцию getindex(строка, символ), чтобы воссоздать строковый метод .index

```
"небоскреб" .индекс( 'c' )
# 4
```

```
getindex( "небоскреб" , "с" )
# 4
```

(U) Используя функцию shout из первого набора базовых упражнений, напишите. shout_words(предложение) функция, которая принимает строковый аргумент и "выкрикивает" каждое слово в отдельной строке.

```
shout_words( "Все любят бананы" )
# ВСЕМ!
# НРАВИТСЯ!
# БАНАНЫ!
```

(U) Напишите функцию extract_longer (длина, предложение), которая принимает длину предложения и слова, затем возвращает список слов предложения, которые превышают заданную длину. Если ни одно из слов не соответствует длине, верните False.

```
extract_longer(5, "Старайтесь не перебивать говорящего". )
# [ 'прерывание', 'динамик'.]
extract_longer(7, "Извините за беспорядок". )
# Ложь
```

Урок 04: Контейнерные типы данных

Обновлено почти 3 года назад [УДАЛЕНО] в COMP 3321 (U) Урок 04: Контейнерные типы данных

НЕКЛАССИФИЦИРОВАНО

(U) Введение

(U) Теперь, когда мы поработали со строками и числами, мы обращаем наше внимание на следующую логическую вещь: контейнеры данных, которые позволяют нам создавать сложные структуры. Существуют разные способы помещения данных в контейнеры, в зависимости от того, что нам нужно с ними делать, и в Python есть несколько встроенных контейнеров для поддержки наиболее распространенных вариантов использования. Встроенные в Python типы контейнеров включают:

1. список
2. кортеж
3. dict
4. set
5. frozenset

(U) Из них tuple и frozenset являются неизменяемыми, что означает, что они не могут быть изменены после их создания, будь то путем добавления, удаления или каким-либо другим способом. Числа и строки также неизменяемы, что должно сделать следующее утверждение более разумным: переменная, которая называет неизменяемый объект, может быть переназначена, но сам неизменяемый объект изменить нельзя.

(U) Чтобы создать экземпляр любого контейнера, мы называем его имя как. функция.иногда известная как. конструктор). Без аргументов мы получаем пустой экземпляр, что не очень полезно для неизменяемых типов. Быстрые способы создания непустых списков, кортежей, определений и даже наборов будут рассмотрены в следующих разделах.

```
list()
dict()
tuple ()
set()
```

(U) Многие встроенные функции и даже некоторые операторы работают с типами контейнеров, где это имеет смысл. Позже мы увидим скрытый механизм, который заставляет это работать; а пока мы перечислим, как это работает, в рамках обсуждения каждого отдельного типа. (U) Список - это упорядоченная последовательность из нуля или более объектов, которые часто бывают разных типов. Обычно он создается путем заключения в квадратные скобки [ ] списка его начальных значений, разделенных запятыми:

```
a = [ "спам" , "яйца" , 5, 3.2, [100, 200, 300]]
фрукт = [ 'Яблоко' , 'Апельсин' , 'Груша' , "Лайм" ]
```

(U) значения могут быть добавлены в список или удалены из него различными способами:

```
фрукт.добавить ( "Банан" )
фрукт.вставка(3, "Вишня" )
фрукты.добавить([ 'Киви' , 'Арбуз' ])
плодоносить.расширять([ "Вишневый" , "Банановый" ])
фрукт.удалить( "Банан" )
фрукты
fruit.pop()
фрукты.pop(3)
фрукты
```

(U) Оператор + работает так же, как метод extend , за исключением того, что он возвращает. новый список.

```
a + фрукт
a
фрукт
```

(U) Другие операторы и методы указывают, как долго. список - это то, есть ли элемент в списке, и если да, то

где и как часто он встречается.

```
len(фрукт)
фрукт.добавить( "Яблоко" )
"Яблоко" внутри фрукты
"Клюква" не в фрукты
фрукты.количество( "Яблоко" )
фрукт.индекс( "Яблоко" ) # Осторожно - может привести к ошибке
fruit.index( "Яблоко" , 1)
```

(U) Понимание списка

(U) Были приложены большие усилия, чтобы упростить работу со списками. Одно из наиболее распространенных применений списка - перебирать его элементы с помощью цикла for, сохраняя результаты каждой итерации в новом списке.

Python удаляет повторяющийся шаблонный код из этого типа процедур с помощью list понятностей. Их лучше всего изучать на примере:

```
a = [i для i в диапазоне (10)]
b = [i**2 для i в диапазоне (10)]
c = [[i, i**2, i**3] для я в диапазоне(10)]
d = [[i, i**2, i**3] для меня диапазоне (10) если i % 2] # Условия !
e = [[i+j для i в 'abcde' ] для j в 'xyz' ] # гнездование!
```

(U) Сортировка и изменение порядка (U) Сортировка - еще одна чрезвычайно распространенная операция со списками. Мы рассмотрим это более подробно позже, но здесь мы рассмотрим самые основные встроенные способы сортировки. Функция sorted работает не только со списками, но всегда возвращает новый список с тем же содержимым, что и у оригинала, в отсортированном порядке. В списках также есть метод сортировки, который выполняет сортировку на месте.

```
фрукт.удалить([ "Киви" , "Арбуз" ]) # не удается сравнить список с str
sorted_fruit = сортированный (фрукты)
sorted_fruit == фрукты
fruit.sort()
sorted_fruit == фрукты
```

(U) Изменение порядка следования списков аналогично, со встроенной функцией reversed и встроенным методом reverse для списков. Обратная функция возвращает итератор, который должен быть преобразован обратно в. список явно. Чтобы отсортировать что-либо в обратном порядке, вы могли бы объединить методы reversed и sorted , но вы должны использовать необязательный аргумент reverse в функциях sorted и сортировка.

```
r_fruit = список(перевернутый(фрукты))
fruit.reverse()
r_fruit == фрукты
отсортированы (r_fruit, reverse=  Верно  )
```

(U) Кортежи

(U) Подобно списку, кортеж представляет собой упорядоченную последовательность из нуля или более объектов любого типа. Они могут быть созданы путем помещения списка элементов, разделенных запятыми, внутрь круглых скобок ( ) или даже путем присвоения списка, разделенного запятыми, переменной вообще без разделителей. Круглые скобки сильно перегружены - они также указывают на вызовы функций и математический порядок операций - поэтому определить одноэлементный кортеж сложно: за одним элементом должна следовать запятая. Поскольку кортеж является неизменяемым, у него не будет ни одного из методов, изменяющих списки, таких как append или sort.

```
a = (1, 2,  "первый и второй"  )
len(a)
отсортирован(a)
a.индекс(2)
a.количество(2)
b =  '1'  ,  '2'  ,  '3'
введите(b)
c_raw =  '1'
c_tuple =  '1'  ,
c_raw == c_tuple
d_raw = (  'd'  )
d_tuple = (  'd'  ,)
d_raw == d_tuple
```

(U) Интерлюдия: обозначение индекса и среза

(U) Для списка упорядоченных контейнеров и кортежа, а также для других упорядоченных типов, таких как строки, часто бывает полезно извлечь или изменить только один элемент или. для этого доступны подмножества элементов, обозначения /ndex и slice . Индексы в Python всегда начинаются с .. Мы начнем с. новый список и работа на примере:

```
животные = [  "тигр"  ,  "обезьяна"  ,  "кошка"  ,  "собака"  ,  "лошадь"  ,  "слон"  ]
животные[1]
животные[1] =  "шимпанзе"
животные[1:3]
животные[3]  у  животных[1:3]
животные[:3]  # начинается с начала
животные[4:]  # идет до конца
животные[-2:]
животные[1:6:2]  # использует необязательный параметр step
```

```
животные[::-1] == список (перевернутый(животные))
```

(U) Потому что возвращается нарезка. новый список, и не только. посмотреть в списке, с его помощью можно сделать. копию.технически. неглубокую копию):

```
same_animals = животные
different_animals = животные[:]
same_animals[0] = "лев"
животные[0]
different_anivals[0] = 'леопард'
different_anivals[0] == животные[0]
```

(U) Словари

(U) dict - это контейнер, который связывает ключи со значениями. Ключи dict должны быть уникальными, и ключами могут быть только неизменяемые объекты. значения могут быть любого типа. (U) В конструкции словаря shortcut используются фигурные скобки { } с двоеточием : между ключами и значениями (например, my_dict = {ключ: значение, ключ: значение1}). Доступны альтернативные конструкторы, использующие ключевое слово dict. значения можно добавлять, изменять или извлекать, используя индексную запись с ключами вместо индексных номеров. Некоторые из операторов, функций и методов, работающих с последовательностями, также работают со словарями.

```
ошибки = {  "муравей": 10,  "богомол"  : 0}
жуки[  "летают" ] = 5
ошибки.обновить({  "паук"  : 1})  # Нравится расширять
удалятьошибки[  "паук"  ]
"летать"  в  жуках
5  в  баги
баги[  "летают" ]
```

(U) В словарях есть несколько дополнительных методов, специфичных для их структуры. Методы, которые возвращают списки, такие как элементы, ключи и значения, не гарантированно выполняют это в каком-либо определенном порядке, но могут быть в согласованном порядке, если в словарь между вызовами не вносится никаких изменений. Метод get часто предпочтительнее индексной записи, потому что он не выдает ошибку, когда запрошенный ключ не найден; вместо этого он возвращает None по умолчанию или значение по умолчанию, которое передается в качестве второго аргумента.

```
ошибки.элементы()       # Список тупиков
ошибки.ключи()
ошибки.значения()
ошибки.получить("летать" )
ошибки.получить("паук"          )
ошибки.получить('spider'          , 4)
ошибки.очистить()
ошибки
```

(U) Наборы и замороженные наборы

(U) Набор - это контейнер, в котором могут храниться только уникальные объекты. Добавление чего-то, что уже есть ничего не даст. но не вызовет ошибки). Элементы набора должны быть неизменяемыми.как ключи в. словаре). Конструкторы set и frozenset принимают любой итерируемый параметр в качестве аргумента, будь то список, кортеж или иное. Фигурные скобки { } вокруг. список значений, разделенных запятыми, можно использовать в Python 2.7 и более поздних версиях в качестве конструктора ярлыков, но это может вызвать путаницу с ярлыком dict . Два набора равны, если они содержат одинаковые элементы, независимо от порядка.

```
числа = набор ([1,1,1,1,1,3,3,3,3,3,3,2,2,3,3,4])
буквы = набор(          "Быстроногая лиса прыгнула поверх хеликобакса"          .ниже())
a = {}          # диета
больше цифр = {1, 2, 3, 4, 5}                    # установить
числа.добавить (4)
числа.добавить(5)
числа.обновить([3, 4, 7])
числа.pop()                  # может быть что угодно
числа.удалить (7)
числа.отбросить (7) без ошибок
```

(U) Замороженный набор создается. аналогичным образом; единственное отличие заключается в изменчивости. Это делает замороженные наборы подходящими в качестве ключей словаря, но замороженные наборы встречаются редко.

```
a = frozenset([1,1,1,1,1,3,3,3,3,32,2,2,3,3,4])
```

(U) Множества принимают обозначение побитовых операторов для операций с множествами, таких как объединение, пересечение и симметричное различие. Это похоже на то, как оператор + используется для объединения списков и кортежей.

```
домашние питомцы = {    "собака" ,    "кошка" ,    "рыба"      }
животные на ферме = {         "корова" ,    "овца"         ,    "свинья" ,    "собака" ,    "кошка"  }
домашние питомцы и животные с фермы          # перекресток
домашние животные | farm_animals             # союз
домашние питомцы                             # симметричная разница
```

```
домашние питомцы - farm_animals                    # асимметричные различия
```

(U) Существуют подробные методы set, которые делают то же самое, но с двумя важными отличиями: они принимают списки, кортежи и другие повторяющиеся элементы в качестве параметров и могут использоваться для обновления набора на месте. Хотя существуют методы, соответствующие всем заданным операторам, мы приведем лишь несколько примеров.

```
farm_animal_list = список(farm_animals) * 2
домашние питомцы.пересечение(farm_animal_list)
домашние питомцы.объединение(farm_animal_list)
домашние питомцы.обновление пересечения(farm_animal_list)
```

(U) Сравнение наборов аналогично: операторы могут использоваться для сравнения двух наборов, в то время как методы могут использоваться для сравнения наборов с другими итерабельными данными. В отличие от чисел или строк, наборы часто несравнимы.

```
домашние питомцы = {  "собака" ,  "кошка" ,  "рыба"  }
farm_animals > домашние питомцы
house_pets < farm_animals
house_pets.intersection_update(farm_animals)
farm_animals > домашние питомцы
house_pets.issubset(farm_animal_list)
```

(U) Coda: больше встроенных функций

(U) Мы видели, как некоторые встроенные функции работают с одним или двумя из этих типов контейнеров, но все из следующих могут быть применены к любому контейнеру, хотя они, вероятно, не всегда будут работать; это зависит от содержимого контейнера. Есть несколько предостережений:

(U) При передаче словаря в качестве аргумента эти функции просматривают ключи словаря, а не значения.

(U) Функции any и all используют логический контекст значений контейнера, например, 0 - это False, а ненулевые числа - True, и все строки истинны, за исключением пустой строки "", которая является False.

(U) Функция sum работает только тогда, когда содержимое контейнера представляет собой числа.

```
generic_container = farm_animals                    # или ошибки, аниматы и т.д.
все(generic_container)
любой (generic_container)
'pig'  в  generic_container
"свинья"  не в  generic_container
длина (generic_container)
макс(generic_container)
мин (generic_container)
```

```
сумма([1, 2, 3, 4, 5])
```

Упражнения на уроке

Упражнение 1. (Задача Эйлера, кратная 3 и 5)

Если мы перечислим все натуральные числа ниже 10, кратные 3 или 5, то получим 3, 5, 6 и 9.

Сумма этих кратных равна 23. Найдите сумму всех чисел, кратных 3 или 5, меньшую 1000.

Упражнение 2.

Напишите функцию, которая принимает список в качестве параметра и возвращает второй список, состоящий из любых объектов,

которые появляются более одного раза в исходном списке

дубликаты ([1,2,3,6,7,3,4,5 > 6]) должны возвращать [3,6]

что должны возвращать дубликаты([, cow' >, pig','goat','horse','pig'])?

Упражнение 3.

Напишите функцию, которая принимает метку части в качестве входных данных и возвращает полную классификацию

convert_classification('U //FOUO') должен возвращать 'НЕСЕКРЕТНО // ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ'

convert_classification ('S //REL TO USA, FVEY') должен возвращать 'СЕКРЕТНО // REL Для США, FVEY'

НЕСЕКРЕТНО

Урок 05: Ввод и вывод файлов

Обновлено почти 2 года назад [УДАЛЕНО] в COMP3321 (U) Урок 05: Ввод и вывод файла

НЕКЛАССИФИЦИРОВАНО

(U) Введение: Становится опасным

(U) Как вы, вероятно, уже знаете, ввод и вывод - это основной инструмент при разработке алгоритмов, и

чтение из файлов и запись в файлы - одна из наиболее распространенных форм. Давайте сразу перейдем к делу, просто чтобы увидеть,

насколько просто записать файл.

```
myfile = открыть(  'data.txt'  ,  'w'  )
мой файл.записать(  "Я записываю данные в свой файл"  )
```

```
myfile.close()
```

(U) И вот оно! Вы можете записывать данные в файлы на Python. Кстати, переменными, которые вы вводите в эту команду open, являются имя файла (в виде строки - не забудьте путь) и режим файла.

Здесь мы записываем файл, на что указывает 'w' в качестве второго аргумента функции open.

(U) Давайте разберем то, что мы на самом деле сделали.

```
открыть( 'data.txt' , 'w' )
```

(U) На самом деле это возвращает нечто с именем. объект file. Давайте назовем это! (U) Опасность: открытие файла, который уже существует для записи, приведет к удалению исходного файла.

```
myfile = открыть( 'data.txt' , 'w' )
```

(U) Теперь у нас есть. переменная для этого файлового объекта, который был открыт в режиме записи. Давайте попробуем записать в файл:

```
myfile.написать( "Я записываю данные в свой файл" )
myfile.read()  # Упс.. . заметили ошибку
myfile.close()  # Угадай, что это сделало ...
```

(U) Есть только несколько файловых режимов, которые нам нужно использовать. Вы видели "w" (написание). Остальные - это "r" (чтение), "a" (добавление), "r + w" (чтение и запись) и "b" (двоичный режим).

```
myfile = открыть ( ' data.txt' , 'r' )
myfile.read()
мой файл.написать( "Я записываю дополнительные данные в свой файл" ) # Упс, еще раз ... проверьте наш режим
mydata = myfile.read()
mydata # ПРИВЕТ! Куда делись данные ....
myfile.close()  # не будь свиньей
```

(U) Отличный способ использовать содержимое файла в блоке - с помощью команды with . Формально это называется. context manager . Неофициально это гарантирует, что файл будет закрыт по окончании блокировки.

```
с помощью open ( 'data.txt' ) как .:
печатать(т.е.читать())
```

(U) Использование with - хорошая идея, но обычно она не является абсолютно необходимой. Python пытается закрыть файлы, когда они больше не нужны. Открытие файлов обычно не является проблемой, если только вы не пытаетесь открыть большое количество файлов одновременно (например, внутри цикла).

(U) Чтение строк из файлов

(U) Вот некоторые из других полезных методов для работы с файловыми объектами:

```
lines_file = открыть( 'fewlines.txt' , 'w' )
файл lines_file.строки записи( "first\n" )
файл lines_file.строки записи( [ "второй \n" , "третий \n" ] )
файл lines_file.close()
```

(U) Аналогично:

```
lines_file = открыть( 'fewlines.txt' , 'r' )
lines_file.readline()
lines_file.readline()
lines_file.readline()
lines_file.readline()
```

(U) И убедитесь, что файл закрыт, прежде чем открывать его снова в следующей ячейке

```
lines_file.close()
```

(U) поочередно:

```
lines = открыть( 'fewlines.txt' , 'r' ).readlines() # Обратите внимание на множественность
строк
```

(U) Примечание: и read, и readline (ы) имеют необязательные аргументы размера, которые ограничивают объем считываемого файла. Для readline (ов) это может возвращать неполные строки. (U) Но что, если файл очень длинный и. вам не нужно или хотите читать их все сразу, файловые объекты ведут себя как их собственный итератор.

```
lines_file = открыть( 'fewlines.txt' , 'r' )
для строки в файле lines_:
печать(строка)
```

Приведенный ниже синтаксис является очень распространенной формулой для чтения файлов. Используйте ключевое слово with, чтобы убедиться, что все идет гладко. Просматривайте файл по одной строке за раз, потому что часто наши файлы содержат одну запись в строке. И делайте что-нибудь с каждой строкой.

```
с помощью open( 'fewlines.txt' ) как мой файл:
для строки в мой файл:
печать(line.strip()) # Функция strip удаляет символы новой строки и пробелы из начала и конца
```

Файл был закрыт после выхода из блока with.

(U) Перемещение с помощью функций tell и seek

(U) Метод tell возвращает текущее положение курсора в файле. Команда поиска устанавливает текущее положение курсора в файле.

```
входной файл = открыть ( 'data.txt' , 'r' )
inputfile.tell()
входной файл.read(4)
входной файл.tell()
входной файл.seek(0)
входной файл.read()
```

(U) Объекты, похожие на файлы

(U) Бывают и другие случаи, когда вам действительно нужно иметь данные в файле (возможно, потому, что другая функция требует, чтобы они были прочитаны из файла). Но зачем тратить время и место на диске, если у вас уже есть данные в памяти? (U) Очень полезный модуль для создания. string into . файлоподобный объект называется stringio . Это займет. string и передаст ему файловые методы, такие как read и write.

```
импорт ввод-вывод
mystringfile = io.StringIO() # Для передачи байтов используйте io.BytesIO
mystringfile.напишите( "Это мои данные!" ) # Мы просто записали в объект, а не в дескриптор файла
mystringfile.read() # Курсор находится в конце!
mystringfile.seek(0)
mystringfile.read()
newstringfile = io.StringIO( "Мои данные" ) # Курсор автоматически будет установлен в положение.
```

(U) Теперь давайте представим, что у нас есть функция, которая ожидает считывания данных из файла, прежде чем работать с ним. Иногда это происходит при использовании библиотечных функций.

```
определить iprintdata (f) :
печатать (f.читать())
iprintdata( 'mydata' ) # Гррр!
my_io = io.StringIO( 'мои данные' )
iprintdata(my_io) # УРА!
```

Упражнения на уроке

Получить данные

Скопируйте сонет из https://urn.nsa.ic.gov/t/tx6qm и вставьте в sonnet.txt.

Упражнение 1.

Напишите функцию с именем file_capitalize(), которая принимает имя входного файла и имя выходного файла, затем записывает каждое слово из входного файла только с заглавной буквы в выходной файл. Убрать все знаки препинания, кроме апострофа.

с заглавной буквы ('sonnet.txt', 'sonnet_caps.txt') => слова, написанные с заглавной буквы в sonnet_caps.txt

Упражнение 2.

Напишите функцию с именем file_word_count(), которая принимает имя входного файла и имя выходного файла, затем записывает каждое слово из входного файла только с заглавной буквы в выходной файл.
Уберите все знаки препинания, кроме апострофа. Все слова в нижнем регистре.

file_word_count(sonnet.txt') => { 'это': 4, 'я': 2, …}

Дополнительный зачет

Запишите словарь подсчетов в файл, один ключ: значение в строке.

НЕСЕКРЕТНО

Урок 06: Среда разработки и инструментарий

Создано более 3 лет назад [УДАЛЕНО] в COMP 3321 (U) Урок 06: Среда разработки и инструментарий

(U) Управление пакетами

(U) Проблема: у Python есть философия "батарейки в комплекте" - у него есть всеобъемлющая стандартная библиотека, но по умолчанию использование других пакетов оставляет желать лучшего:

- В Python нет пути к классам, и если вы не являетесь пользователем root, вы не сможете устанавливать новые пакеты для всей системы.
- Как вы можете поделиться скриптом с кем-то еще, если вы не знаете, какие пакеты установлены в их системе?

Иногда вам приходится использовать Project A, который полагается на пакет, требующий awesome-package v.1.1, но вы пишете Project B и хотите использовать некоторые функции, которые являются новыми в awesome-package v.2.0?

Лучшего в своем классе менеджера пакетов нет в стандартной библиотеке Python.

(U) Решение: virtualenv

(U) Пакет virtualenv создает виртуальные среды, то есть изолированные пространства, содержащие собственные экземпляры Python. Он предоставляет служебный скрипт, который управляет вашей средой для активации выбранной вами среды. (U) Он уже установлен и доступен на виртуальной машине класса. Флаг -p указывает, какой исполняемый файл Python использовать в качестве основы для виртуальной среды:

```
[[УДАЛЕНО] ~]$ virtualenv NEWENV.p.usr/local/bin/python
Новый исполняемый файл python в NEWENV/bin/python
Установка Setuptools..... Выполнено
Установка Pip..... Выполнено
[[УДАЛЕНО] ~]$ какой python
/usr/local/bin/ python
[[УДАЛЕНО] ~]$ исходный NEWENV/bin/активировать
(NEWENV)[[УДАЛЕНО] ~]$ какой python
~/NEWENV/bin/ python
(NEWENV)[[УДАЛЕНО] ~]$ исходный NEWENV/bin/ деактивировать
[[УДАЛЕНО] ~]$
```

(U) Пакет virtualenv можно загрузить и запустить как скрипт для создания виртуальной среды на основе любой недавней установки Python. В виртуальной среде предварительно установлен менеджер пакетов pip, который можно подключить к внутреннему зеркалу индекса пакетов Python (PyPI), экспортировав правильный адрес в переменную среды PIP_INDEX_URL:

```
[[УДАЛЕНО] ~]$ echo.PIP_INDEX_URL
http://bbtux022.gp.proj.nsa.ic.gov/PYPI
[[УДАЛЕНО] ~]$ python
python 2.7.5 (по умолчанию, 6 ноября 2013 г., 10:23:48)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] на linux2
Введите "справка" , "авторскиеправа" , "кредиты" или "лицензия" для получения дополнительной информации.

импорт запрашивает
обратную трассировку.последний вызов):
Файл "<стандартный код>", строка., в .module>
ImportError: Ни один модуль не запрашивает с именем
exit()

[[УДАЛЕНО] ~]$ исходный файл NEWENV/bin/активировать
[[УДАЛЕНО] ~]$ исходный файл NEWENV/bin / активировать
```

(NEWENV)[[УДАЛЕНО] ~] запросы на установку $ pip

Запросы на загрузку / распаковку

Запросы на загрузку-2.0.0.tar.gz.362kB): загружено 362 КБ

Выполняется setup.py egg_info для запросов на отправку

Установка собранных пакетов: запросы

Выполняется setup.py установка для запросов

Успешно установленные запросы на

Очистку...

(NEWENV) [[УДАЛЕНО] ~]$ python

Python 2.7.5 (по умолчанию, 6 ноября 2013 г., 10:23:48)

[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] на linux2

Введите "справка" , "авторскиеправа" , "кредиты" или "лицензия" для получения дополнительной информации.

импортируйте запросы

requests._version_

'2.0.0'

импортируйте систему

sys. путь

[' ', '/главная страница/[УДАЛЕНО]/NEWEW/lib/python27.zip >', '/home/NEWENV/lib/python2.7', ' /home/lib/NEMENV/l ib/pytht.n

_, _. —f.NEWENV/lib/python2.7/lib-tk., '/home/ I. /NEMENV/lib/python2.7/lib-old', '/home/ I. /NEMENV/li

b/python2.7/lib-dynJLbad', 1.usr /local/lib/python2.7', ' /usr/ local/lib/python2.7/plat-linux2., '/usr/ local/li

k', '/home/r./ NEWENV/lib/ pythpn2.7/site-packages']

exit()

(NEMENV) [[УДАЛЕНО] ~] замораживание $ pip

запросы ==2.0.0

wsgiref ==0.1.2

** Теперь у нас есть место для установки пользовательского кода и способ поделиться им! **

Разработайте код внутри ~/NEWENV/lib/python2.7/site-packages

Захватите установленные пакеты с помощью pip freeze >> requirements.txt и установите их на новый

virtualenv с помощью pip install -r requirements.txt

(U) Абсолютный пакет услуг

(U) IPython - это альтернативная интерактивная оболочка для Python с множеством интересных функций, среди которых:

завершение работы с вкладками,

цветной вывод,

просмотр богатой истории,

улучшенный интерфейс справки,

"волшебные" команды,

веб-интерфейс ноутбука с удобным доступом к файлам и

распределенные вычисления.не спрашивайте об этом) (U) Для начала:

(NEWENV) python pip устанавливает ipython

Загрузка ipython-1.1.0.tar.gz.8*.7MbJ8.7AB * загружено ■

...

Успешно установленный ipython

Очищает ...

(NEWENV) [[УДАЛЕНО] ~]$ ipython

Python 2.7.5 (по умолчанию, 6 ноября 2013, 10:23:48)
"авторское право" , "кредиты" или "лицензия" для получения дополнительной информации.
Введите
IPython 1.1.0 - улучшенный интерактивный Python.

? -> Введение и обзор возможностей IPython .

% краткая ссылка -> Краткая справка.

справка -> Python' .
возражать? -> Сведения о.object ', использовать.object??' для дополнительных сведений.

In.1]: Является

BASE3/ Hello World.html Привет, мир.ipynb NEWENV/

B[2]: hist

- это

hist

B[3]: импорт os

B[4]: os.path.нажмите tab

os.путь os.pathconf os.pathconf_pames os.pathsep

B.4]: os.path

(U) Чтобы использовать веб-интерфейс, вам необходимо установить дополнительные пакеты: (NEWENV) [[УДАЛЕНО] ~]$ pip install pyzmq tornado jinja2 pygments (NEWENV) [[УДАЛЕНО] ~]$ ipython * notebook --no-mathjax (U) Для получения потрясающей встроенной графики требуются еще два пакета (NEWENV) [[УДАЛЕНО] ~]$ pip install numpy HELLO NSA (NEWENV) [[УДАЛЕНО] ~]$ pip install matplotlib

Урок 07: Объектное ориентирование: использование классов

Обновлено 9 месяцев назад [УДАЛЕНО] в COMP 3321 (U) Введение в классы, объекты и наследование в Python.

НЕКЛАССИФИЦИРОВАНО

(U) Введение

(U) Из названия вы можете видеть, что объектно-ориентированное программирование изобилует абстракцией и усложнением. Мужайтесь: не нужно бояться или избегать объектно-ориентированного программирования в Python! Это просто еще один простой в использовании, гибкий и динамичный инструмент в deep toolbox, который делает доступным Python . Фактически, мы используем объекты и объектно-ориентированные концепции с тех пор, как написали первую строку кода Python, так что это уже знакомо. В этом уроке мы задумаемся более глубоко

о том, чем мы занимались все это время и как мы можем воспользоваться преимуществами этих идей.

(U) Рассмотрим, например, разницу между. функция и. способ:

```
название = "Отметить"
len( имя ) # функция
имя .upper() # метод
```

(U) В этом примере name - это экземпляр типа str. Другими словами, name - это объект этого типа. Объект - это просто удобная оболочка. комбинация некоторых данных и функциональных возможностей, связанных с этими данными, воплощенных в методах. До сих пор вы, вероятно, думали о каждом str только в терминах его данных, то есть буквальной строки "Mark", которая использовалась для присвоения переменной. Методы, которые работают с name, были определены только один раз, в определении класса, и применяются к каждой строке, которая когда-либо создавалась. Методы на самом деле - это то же самое, что и функции, которые живут внутри класса, а не вне его. (Этот абзац, вероятно, все еще кажется действительно запутанным. Попробуйте перечитать это в конце урока!)

(U) Твой первый урок

(U) Так же, как ключевое слово def используется для определения функций, ключевое слово class используется для определения типа объекта, который будет генерировать новый тип объекта, которому вы можете дать имя!. В качестве текущего примера, мы будем работать с. класс, для которого мы выберем имя Person :

```
класс Лицо (объект):
    пропуск
тип (Персона)
тип(Person) == тип (int)
никто = Person()
тип (никто)
```

(U) Сначала класс Person мало что делает, потому что он абсолютно пустой! Это не так бесполезно, как кажется, потому что, как и все остальное в Python, классы и их объекты динамичны.
(объект) после Person не является вызовом функции; здесь он называет родительский класс. Несмотря на то, что индивидуальные занятия кажутся скучными, в них есть основы:

- класс Person - это такой же класс, как int или любой другой встроенный,
- мы можем создать экземпляр, используя имя класса as. функция-конструктор, и
- тип экземпляра nobody - Person, точно так же, как тип(1) - int. (U) Поскольку это практически все, что мы можем сделать, давайте начнем сначала и внедрим некоторые данные и функциональность в Person :

```
класс Человек . объект ):
    вид = "Homo sapiens"
    определение говорить(от себя):
        Возврат "Привет, как дела?"
```

```
никто = Person()
никто.вид
никто.разговор()
```

(U) Очень важно предоставить любому методу (т. е. функции, определенной в классе) хотя бы один аргумент, который почти всегда называется self . Это потому, что внутренне Python переводит nobody.talk() во что- то вроде Person.talk(никто) . (U) Давайте поэкспериментируем с классом Person и его объектами и сделаем такие вещи, как переназначение других атрибутов данных.

```
некто = Персона()
некто.вид =                    "Homo internetus"
somebody.name =          "Отметить"
никто.вид
Person.species =              "Неизвестно"
никто.вид
некто.вид
Person.name =           "Неизвестный"
никто.имя
кем-то.название
дель    кто-то.название
кто-то.название
```

(U) Хотя мы могли бы добавлять имя к каждому экземпляру сразу после его создания, по одному за раз, не было бы неплохо присвоить подобные атрибуты для конкретного экземпляра при первом создании объекта? Функция init позволяет нам сделать это. За исключением забавных подчеркиваний в названии, это всего лишь обычная функция; мы даже можем присвоить ей аргументы по умолчанию.

```
класс Person.object ):
        вид = "Homo sapiens"
        def __init__.self, имя="Неизвестно", возраст=18):
                самость. имя = name
                самость. возраст = age
        определение (для себя):
                Возврат.Привет, меня зовут {}." .format(self.name )
отметка = Персона("Отметка", 33)
generic_worker = Персона()
generic_worker = Персона (возраст=41)
generic_worker.возраст
generic_worker.name
```

(U) В Python нет ничего необычного в прямом доступе к атрибутам объекта, в отличие от некоторых языков (например, Java), где это считается плохой формой и все делается с помощью методов получения и установки . Это связано с тем, что в Python атрибуты можно добавлять и удалять в любое время, поэтому средства получения и установки могут оказаться бесполезными к тому времени, когда вы захотите их использовать.

```
отметить.favorite_color =  "зеленый"
del  generic_worker.name
generic_worker.name
```

(U) Одним из потенциальных недостатков является то, что в python нет реального эквивалента частных данных и методов; каждый может видеть все. Существует вежливое соглашение: предполагается, что другие разработчики должны рассматривать атрибут как закрытый, если его имя начинается с. одиночный символ подчеркивания (_). И еще есть хитрость: имена, которые начинаются с двух символов подчеркивания (_), искажаются, чтобы затруднить доступ к ним. (U) Метод init - это лишь один из многих, которые могут помочь вашему классу вести себя как полноценный встроенный в Python объект . Чтобы контролировать, как печатается ваш объект, реализуйте str , а чтобы контролировать, как он выглядит в качестве выходных данных интерактивного интерпретатора, реализуйте repr . На этот раз мы не будем начинать с нуля; мы добавим их динамически.

```
определение person_str(self ):
    return.Name: {0}, Возраст: {1}" .формат (self. имя, фамилия, возраст)

Person._str_ = person_str

определение person_repr(self ):
    Возврат.Персона('{0}',{1})" . формат (self.name, я. возраст)

Человек._repr_ = person_repr

печать (пометка) # какой специальный метод печати используется?
отметьте # какой специальный метод Jupyter использует для автоматической печати?
```

(U) Найдите минутку, чтобы подумать о том, что только что произошло:

Мы добавили методы в. класс после создания. куча объектов, но каждый объект в этом классе сразу же смог использовать этот метод.
Поскольку это были специальные методы, мы могли сразу же использовать встроенные функции Python.как str) для этих объектов. (U) Будьте осторожны при реализации специальных методов. Например, вы можете захотеть, чтобы сортировка класса Person по умолчанию основывалась на возрасте. Специальный метод it(self,other) будет использоваться Python вместо встроенной функции lt даже для сортировки. (Python. вместо этого использует cmp.) Несмотря на то, что это просто, это проблематично, потому что это заставляет объекты казаться равными, когда они просто одного возраста!

```
определение person_eq  (я, другой):
    возвращение себя.возраст == другой.возраст
Человека.__eq__ = person_eq
боб = Персона (  "Боб"  , 33)
боб == марк
```

(U) В подобной ситуации, возможно, было бы лучше реализовать подмножество расширенного сравнения

методы, может быть, только it и gt, или используйте более сложную функцию эквалайзера, которая способна однозначно идентифицировать все объекты, которые вы когда-либо создадите. (U) Хотя мы показали примеры добавления методов в класс постфактум, обратите внимание, что на практике это редко делается таким образом. Здесь мы сделали это просто для удобства, чтобы не приходилось заново определять класс каждый раз, когда мы хотим создать новый метод. Обычно вы просто определяете все методы класса в самом классе. Если бы мы должны были сделать это с методами a str, repr и eq для класса Person выше, классу понравилось бы следующее:

```
класс Лицо (объект):
    вид = "Homo sapiens"
    определение __init__(я, имя= "Неизвестно", возраст=18):
        самость. имя = name
        самость. возраст = age
    определение говорить(от себя) :
        Возврат "Привет, меня зовут {}". .формат (self . имя)
    def __str__ (самостоятельно):
        Возврат "Имя: {0}, Возраст: {1}" .формат (self.name, self. age)
    определение __repr__ (самостоятельно):
        Возврат "Человек('{0}',{1})" .формат (self.name, self. age)
    определение __eq__(я, другой):
        вернуться к себе. возраст == другой.возраст
```

Наследование

(U) Существует много типов людей, и каждый тип может быть представлен своим собственным классом. Было бы неприятно, если бы нам пришлось переопределять фундаментальные черты личности в каждом новом классе. К счастью, наследование дает нам способ избежать этого. Мы уже видели, как это работает: Person наследует от класса object (или является подклассом). Однако любой класс может быть унаследован от него (т.е. Иметь потомков).

```
класс Студенты Человек ):
    пора спать = 'midnight'
    def do_homework (самостоятельно):
        импорт время
        печать( - Мне нужно работать. )
        время ожидания(5)
        печать( - Неужели... только что заснул? )
тайлер = Студент( "Тайлер" , 19)
тайлер.виды
тайлер.говорить()
тайлер.делать домашнюю работу()
```

(U) Объект из подкласса обладает всеми свойствами родительского класса вместе с любыми дополнениями из его собственного определения класса. Вы все еще можете легко переопределить поведение родительского класса - просто создайте метод с таким же именем в подклассе. Использовать поведение родительского класса в дочернем классе сложно, но весело, потому что вам нужно использовать суперфункцию.

```
класс Служащий . Человек ):
    защита говорить(от себя):
        talk_str = супер.Сотрудник, самостоятельно).talk()
        возврат talk_str + "Я работаю на {)" .формат (self. работодатель)
фред = Сотрудник( "Фред Флинстоун" , 55)
фред.работодатель = "Компания по производству сланцевых пород и гравия"
fred.talk()
```

(U) Синтаксис здесь на первый взгляд странный. Суперфункция принимает класс (то есть тип) в качестве своего первого аргумента, а объект, происходящий от этого класса, - в качестве второго аргумента. Объект имеет цепочку классов-предков. Для fred эта цепочка равна [Employee, Персона, объект]. Суперфункция проходит по этой цепочке и возвращает класс, следующий за классом, переданным в качестве первого аргумента функции. Следовательно, super можно использовать для пропуска вверх по цепочке, пропуская модификации, сделанные в промежуточных классах. (U) В качестве второго, более распространенного (но более сложного) примера, часто полезно добавлять дополнительные свойства к объектам подкласса в конструкторе.

```
класс Служащий (Лицо):
    def __init__ (имя, отчество, возраст, место работы):
        супер(Сотрудник, самоучка). __init__(имя, возраст)
        self.employer = работодатель
    определить говорить(self):
        talk_str = супер (Сотрудник, я сам).talk()
        возврат talk_str + " Я работаю на {}" .форматировать(self. работодатель)
фред = Сотрудник ( "Фред Флинстоун" , 55 лет, "Компания по производству сланцевых пород и гравия" )
фред.поговори()
```

(U) Класс в Python может иметь более одного указанного предка.это иногда называется полиморфизмом). Мы не будем вдаваться здесь в подробности, за исключением указания на то, что он существует и является мощным, но сложным.

```
класс StudentEmployee.Студент, Служащий):
    pass
ann = StudentEmployee("энн", 58, "Семейные службы")
ann.talk()
билл = StudentEmployee("билл", 20) # что здесь происходит? почему?
```

(U) Упражнения на уроке

(U) Упражнение 1.

(U) Напишите класс запроса, который имеет следующие атрибуты:

- классификация

обоснование

селектор (U) Предоставляет значения по умолчанию для каждого атрибута. рассмотрите возможность использования None). Сделайте это так, чтобы

при печати вы могли красиво отобразить все атрибуты и их значения.

```
# ваш класс определение здесь
```

(U) После этого должно сработать что-то вроде этого:

```
query1 = Запрос( "TS//SI//ОТНОСИТЕЛЬНО США, FVEY" , "Основной адрес электронной почты Zendian diplomat" , "ileona@stato.gov.
печать(запрос 1)
```

(U) Упражнение 2.

(U) Создайте класс RangedQuery, который наследуется от Query и имеет дополнительные атрибуты:

дата начала

дата окончания (U) Пока просто введите даты в форме ГГГГ-ММ-ДД. Пока не беспокойтесь о дате

форматировании или проверке ошибок. Мы поговорим о модуле datetime и обработке исключений

позже. (U) Укажите значения по умолчанию для этих атрибутов. Убедитесь, что вы включили инициализатор класса Query

в инициализатор RangedQuery . Убедитесь, что новый класс также можно

красиво распечатать.

```
# ваш класс определение здесь
```

(U) После этого это должно сработать:

```
query2 = RangedQuery( "TS //SI//REL В США, FVEY" , "Основной IP-адрес Zendian diplomat" , "10.254.18.1
печать(запрос 2)
```

(U) Упражнение 3.

(U) Измените класс запроса на accept. список селекторов, а не. одиночный селектор. Убедитесь, что вы

по-прежнему можете печатать все нормально.

НЕКЛАССИФИЦИРОВАНО

Урок 07: Дополнение

Обновлено 11 месяцев назад [УДАЛЕНО] в приложении COMP 3321 (U) к уроку 07 на основе упражнений из предыдущих лекций. Возможно, вы написали. используйте подобную функцию, чтобы проверить, есть ли товар в вашем списке покупок, и напечатать что-нибудь язвительное, если его нет:

```
def in_my_list (предмет) :
    mylist = [ 'яблоки' , 'молоко' , 'масло' , "апельсиновый сок" ]
    если товар в моем списке:
        Возврат "Понял!"
    остальное
        Возврат "Не-а!"
in_my_list( 'яблоки' )
in_my_list( "шоколад" )
```

Но что, если бы я действительно хотела, чтобы шоколад был в моем списке? Мне пришлось бы переписать свою функцию. Если бы я написал класс вместо функции, я смог бы изменить свой список.

```
класс My_list (объект) :
my_list = [ 'яблоки' , 'молоко' , 'масло' , "апельсиновый сок" ]
    определение in_my_list (self, item):
        если товар в self.my_list:
            Возврат "Понял!"
        остальное
            Возврат "Не-а!"

december = My_list()
december.in_my_list( "шоколад" )
december.my_list.декабрь.my_list +[ "шоколад" ]
december.in_my_list( "шоколад" )
```

Теперь у меня есть хороший шаблон для списков покупок и поведения в списке покупок

```
jan = My_list()
december.my_list
jan.my_list
```

Это бесполезно:

```
печать (декабрь)
```

Итак, мы перезаписываем функцию str, унаследованную нами от object:

```
класс My_list (объект):
    my_list = [ 'яблоки' , 'молоко' , 'масло' , "апельсиновый сок" ]
    def ___str__ (самостоятельно):
```

```
        Возврат ( 'Мой список: {}' .format( ', ' . join(self.my_list))
    определение __repr__ (самостоятельно):
        вернуть self.__str__()
    def in_my_list (self, item):
        если товар в self.my_list:
            Возврат "Понял!"
        остальное
            Возврат "Не-а!"

декабрь = My_list()
печать (декабрь)

декабрь
```

Может быть, я также хочу, чтобы мне было легче проверить, есть ли в списке моя любимая закуска ...

```
класс My_list (объект):
    my_list = [ 'яблоки' , 'молоко' , 'масло' , "апельсиновый сок" ]
    определение __инициализация__(self, перекус= "шоколад" ):
        self.snack = перекус
    def ___str__ (самостоятельно):
        Возврат ( 'Мой список: {}' . format ( ', ' .join.self.my_list))
    определение in_my_list (self, item):
        если товар в self.my_list:
            Возврат "Понял!"
        остальное
            Возврат "Не-а!"
    защита snack_check (самостоятельно):
        возврат self. перекус в self.my_list
#Мой любимый перекус - шоколад … Но в январе. я собираюсь притвориться, что это апельсины
jan = My_list( "яблоки" )
января.snack_check()
#Но в феврале я возвращаюсь к значению по умолчанию
февраль = My_list()
февраль.snack_check()
```

Об этом предмете...

```
режиссер (объект)
```

Это все то, что вы наследуете, создавая подкласс object.

```
класс список заглавных(Мой список) :
    определение in_my_list (self.item):
        ответ = super(caps_list,self).in_my_list(элемент)
        возврат ответ.upper()
```

```
shouty = caps_list()
shouty.in_my_list( "шоколад" )
каталог(caps_list)
```

Вы также можете вызвать суперкласс напрямую, вот так:

```
класс список заглавных(Мой список) :
    определение in_my_list (self.item):
        # Но вам все равно придется передать seif
        response = My_list(in_my_list.self, элемент)
        вернуть response.upper()
shouty = caps__list()
shouty.in_my_list( 'шоколадный' )
```

Super на самом деле предполагает правильные вещи… Большую часть времени.

```
класс список заглавных(Мой список) :
    определение in_my_list (self.item):
        response = super().in_my_list(элемент)
        вернуть response.upper()

shouty = caps_list()
shouty.in_my_list( 'шоколадный' )
помощь(супер)
```

Урок 08: Модули, пространства имен и пакеты

Обновлено более 2 лет назад [УДАЛЕНО] в COMP 3321 (U //FOUO) Уроком по модулям Python, пространствам имен и пакетам для COMP3321.

НЕСЕКРЕТНО//ТОЛЬКО ДЛЯ ОФИЦИАЛЬНОГО ИСПОЛЬЗОВАНИЯ

(U) Модули, пространства имен и пакеты

(U) Мы уже достаточно использовали модули. бит - фактически, каждый раз, когда мы запускаем импорт. Но что такое. именно модуль?

(U) Мотивация

(U) При работе в Jupyter вам не нужно беспокоиться о том, что ваш код исчезнет при выходе. Вы можете сохранить записную книжку и поделиться ею с другими. Записная книжка Jupyter ведет себя как скрипт на Python: текстовый файл, содержащий исходный код на Python. Вы можете передать этот файл python

интерпретатор в командной строке и выполните весь код в файле (что-то вроде "Run All" в записной книжке Jupyter):

```
$ python awesome.py
```

(U) Однако существует несколько существенных ограничений на совместное использование кода в записных книжках Jupyter:

1. что делать, если вы хотите поделиться с кем-то, у кого установлен python, но нет Jupyter?
2. что, если вы хотите поделиться частью кода с другими (или повторно использовать часть его самостоятельно)?
3. что, если вы пишете большую, сложную программу? (U) У всех них есть собственные решения в Jupyter:
4. преобразуйте notebook в. script (Файл> Загрузить как> Python)
5. скопируйте-вставьте ...?
6. сделайте большой, беспорядочный блокнот ...? (U) ... но они быстро становятся громоздкими. Вот тут-то и появляются модули .

(U) Модули

(U) По сути, модуль в Python - это просто другое название скрипта. Это просто файл, содержащий определения и инструкции Python. Имя файла - это имя модуля, за которым следует расширение .py . Однако, как правило, мы не запускаем модули напрямую - мы импортируем их определения в наш собственный код и используем их там. Модули позволяют нам писать модульный код, организуя нашу программу в логические блоки и помещая эти блоки в отдельные файлы. Затем мы можем поделиться и повторно использовать эти файлы по отдельности как части других программ.

(U) Стандартные модули

(U) Python поставляется с библиотекой стандартных модулей, так что вы можете продвинуться довольно далеко, не создавая свой собственный. Мы уже видели некоторые из этих модулей, и большая часть следующей недели будет посвящена изучению полезных модулей. Они полностью задокументированы в справочнике по стандартной библиотеке Python . (U) Потрясающий пример (U) Чтобы лучше понимать модули, давайте создадим наши собственные. Это добавит немного кода на Python. вызываемый файл awesome.py в текущем каталоге.

```
содержание =  '''
класс awesome(объект):
    def __init__(self, awesome_thing):
        self.thing = потрясающая вещь
    def __str__(self):
        возвращает "{0.thing} потрясающе!!!".format(self)

a = Потрясающе ("Все")
вывести(а)
```

```
'''
с открытым ( 'awesome.py' , "w" ) как f:
    f.записать (содержимое)
```

(U) Теперь вы можете запускать python awesome.py в командной строке в виде скрипта Python.

(U) Использование модулей: импорт

(U) Вы также можете импортировать awesome.py сюда как. module:

```
импорт потрясающий
```

(U) Обратите внимание, что вы не указываете расширение файла при его импорте. Python знает, что нужно искать. файл в вашем пути называется awesome.py . (U) При первом импорте модуля Python выполняет код внутри него. Любые определенные функции, классы и т.д. будет доступен для использования. Но обратите внимание, что происходит при повторной попытке импортировать его:

```
импорт потрясающий
```

(U) Предполагается, что другие инструкции (например, назначение переменных, печать) предназначены для помощи в инициализации модуля. Вот почему модуль запускается только один раз. Если вы попытаетесь импортировать один и тот же модуль дважды, Python не будет повторно запускать код - он будет ссылаться на уже импортированную версию. Это полезно, когда вы импортируете несколько модулей, которые, в свою очередь, импортируют один и тот же модуль. (U) Однако, что, если модуль изменился с момента вашего последнего импорта, и вы действительно хотите повторно импортировать его?

```
contents = '''
класс Awesome(объект):
    def __init__ (self, awesome_thing):
        self.thing = awesome_thing
    def __str__(self):
        вернуть "{0.вещь} потрясающая !!!".формат (self)
    определение классности (group):
        верните "Все круто, когда ты часть {0}".формат (группа)

a = Потрясающе ("Все")
выведите(a)
'''
с помощью открыть ( 'awesome.py' , 'w' ) как .:
f.запись(содержимого)
```

(U) Вы можете внести новую версию с помощью модуля importlib:

```
импорт importlib
importlib.перезагрузить (потрясающе)
```

(U) Вызов кода модуля

(U) Основная точка импорта. модуль предназначен для того, чтобы вы могли использовать определенные им функции, классы, константы, и т.д. По умолчанию мы получаем доступ к объектам, определенным в модуле awesome, добавляя к ним префикс с именем модуля.

```
печать (потрясающая.Потрясающе ( "Нобелевская премия" ) )
потрясающе.круто ( "команда" )
печать(потрясающая.а)
```

(U) Что, если мы устанем постоянно писать "потрясающе"? У нас есть несколько вариантов.

(U) Использование модулей: импортируйте ___ как ____

(U) Во-первых, мы можем выбрать псевдоним для модуля:

```
импорт awesome как awe
печать (awe.Удивительный ( "Книга греческих древностей" ) )
awe.cool ( "сообщество разработчиков python" )
печать(awe.a)
```

(U) Использование модулей: из __ import __

(U) Во-вторых, мы можем импортировать определенные вещи из модуля awesome в текущее пространство имен:

```
от awesome импорт круто
круто( "этот класс" )
печать(Потрясающая( "Обрывок веревки" )) # сработает ли это?
вывести(а) # сработает ли это?
```

(U) Получить все:

```
От __ импорт *
```

(U) Наконец, если вы действительно хотите импортировать все из модуля в текущее пространство имен, вы можете сделать это:

```
от awesome импорт * # БУДЬ ОСТОРОЖЕН
```

(U) Теперь вы можете повторно запустить указанные выше ячейки и заставить их работать. (U) Почему вам может потребоваться быть осторожным с этим методом?

```
# что, если бы вы определили это до импорта?
def круто ():
    возврат "Что-то важное - это довольно круто"
круто()
```

(U) Возьмите что-нибудь одно и переименуйте: из __ import __ в ___

(U) Вы можете использовать как из, так и как, если вам нужно:

```
из потрясающе импортировать круто как coolgroup
cool()
coolgroup( "команда" )
```

(U) Приведение в порядок с помощью main

(U) Помните, как он печатал что-то обратно, когда мы запускали import awesome ? Нам не нужно, чтобы это распечатывалось каждый раз, когда мы импортируем модуль. (И на самом деле не инициализирует ничего важного.) К счастью, Python предоставляет способ различать запуск файла как скрипта и импорт его как модуля, проверяя специальное имя переменной. Давайте снова изменим код нашего модуля:

```
содержимое = '''
класс awesome(объект):
    def __init__ (самооценка, потрясающая вещь):
        self.thing = потрясающая вещь
    def __str__ (self):
        вернуть "{0.вещь} потрясающая !!!".формат (self)
    определение классности (group):
        возвращает "Все круто, когда ты часть {0}".формат (группа)

if _name_ == '__main__
    a = Потрясающе ("Все")
    вывести(а)
'''
с помощью open ( 'awesome.py' , "w" ) как f:
    f.записать (содержимое)
```

(U) Теперь, если вы запустите модуль как. скрипт из командной строки, он создаст и распечатает пример

из Потрясающего класса. Но если вы импортируете его как модуль, этого не произойдет — вы просто получите определение класса и функции.

```
importlib.перезагрузка(потрясающе)
```

(U) Волшебство здесь в том, что name - это имя текущего модуля. Когда вы импортируете модуль, его именем является название модуля (например, awesome), как и следовало ожидать. Но запущенный скрипт (или записная книжка) также использует специальный модуль на верхнем уровне под названием main : name (U), поэтому, когда вы запускаете модуль непосредственно как скрипт (например, python awesome.py), его имя на самом деле является main, а больше не именем модуля . (U) Это обычное соглашение для написания скрипта на Python: организуйте его так, чтобы его функции и классы могли быть импортированы чисто, и поместите "склеивающий" код или поведение по умолчанию, которое вы хотите, когда скрипт запускается непосредственно под проверкой имени. Иногда разработчики также помещают код в функцию с именем main() и вызывают ее вместо этого, вот так:

```
def main ():
    a = Потрясающий( "Все" )
    вывести(a)

если _name_ == '__main__' :
    main()
```

(U) Пространства имен

(U) В Python пространства имен - это то, что хранит имена всех переменных, функций, классов, модулей и т.д., используемых в программе. Пространства имен ведут себя как большой словарь, который сопоставляет имя названному объекту. (U) Два основных пространства имен - это глобальное пространство имен и локальное пространство имен. Глобальное пространство имен доступно отовсюду в программе. Локальное пространство имен будет меняться в зависимости от текущей области видимости - находитесь ли вы в функции, цикле, классе, модуле и т.д. Помимо локального и глобального пространств имен, каждый модуль имеет свое собственное пространство имен.

(U) Глобальное пространство имен

(U) функция dir() без аргументов фактически показывает вам имена в глобальном пространстве имен.

```
режиссер()
```

(U) Другой способ увидеть это - с помощью функции globals( ), которая возвращает. словарь не только названий, но и их значений.

```
отсортировано(глобальные().ключи())
dir() == отсортировано(глобальные().ключи())
```

```
globals()[ "потрясающий" ]
globals()[ "круто" ]
globals()[ 'coolgroup' ]
```

(U) Локальное пространство имен

(U) К локальному пространству имен можно получить доступ с помощью locals() , которое ведет себя точно так же, как globals() . (U) Прямо сейчас локальное пространство имен и глобальное пространство имен совпадают. Мы находимся на верхнем уровне нашего кода, а не внутри. функция или что-то еще.

```
глобальные () == локальные()
```

(U) Давайте взглянем на это с другой точки зрения. звук / пакет верхнего уровня init.py Инициализируйте форматы / подпакет звукового пакета для преобразования форматов файлов init.py wavread.py wavwrite.py aiffread.py aiffwrite.py эффекты / Подпакет для звуковых эффектов init.py echo.py surround.py reverse.py фильтры / Подпакет для фильтров init.py equalizer.py vocoder.py karaoke.py

(U) Вы можете получить доступ к подмодулям, объединив их в цепочку с помощью точечных обозначений:

```
импорт звук.эффекты.реверс
```

(U) Также существуют другие способы импорта работ:

```
от звука.фильтры импорт караоке
```

(U) init.py

(U) Что это за особенный файл init.py?

- (U) Его присутствие требуется, чтобы сообщить Python, что каталог является пакетом
- (U) Он может быть пустым, пока он там есть
- (U) Обычно он используется для инициализации пакета (как следует из названия) (U) init.py может содержать любой код, но лучше всего, чтобы он был коротким и фокусировался только на том, что необходимо для инициализации пакета и управления им . Например:
- (U) установка переменной all, указывающей Python, какие модули включать, когда кто-то запускает **из package import ***
- (U) автоматически импортирует некоторые из подмодулей, чтобы, когда кто-то запускает import package , он мог запускать package.function, а не package.submodule.function

(U) Установка пакетов

(U) Пакеты на самом деле являются распространенным способом совместного использования и распространения модулей. Пакет может содержать. один модуль - нет необходимости в том, чтобы он содержал несколько модулей. Если вы хотите работать с. Модуль Python, которого нет в стандартной библиотеке (т. Е. Не установлен с Python по умолчанию), тогда вам, вероятно, потребуется установить пакет, который его содержит. Разработчики Python обычно не делятся файлами отдельных модулей и не устанавливают их.

(U) pip и PyPI

(U) В командной строке стандартным инструментом для установки пакета является pip, менеджер пакетов Python . (в настоящее время pip поставляется с Python по умолчанию, но если вы используете более старую версию, вам, возможно, придется установить ее самостоятельно.) Чтобы использовать pip, вам нужно настроить его так, чтобы он указывал на. репозиторий пакетов. Снаружи большой репозиторий, которым пользуются все, называется PyPI (он же Сырный магазин).

(U //FOUO) РЕПОРТЕР и сотрудник АНБ

(U / / FOUO) REPOMAN также импортирует и размещает. зеркало PyPI на верхней стороне. Кроме того, есть . сервер nsa-pip, который подключается как к зеркалу PyPI REPOMAN, так и к. множество внутренних пакетов, разработанных NSA, размещенных в GitLab.

(U / / FOUO) Список внутренних пакетов АНБ

(U //FOUO) Ссылки на некоторые пакеты АНБ. электронные документы

(U) ipydeps & pypki2

(U // FOUO) Если вы работаете в ноутбуке Jupyter, может быть неудобно пытаться устанавливать пакеты из командной строки с помощью pip, а затем использовать их. Вместо этого ipydeps - это модуль, который позволяет вам устанавливать пакеты непосредственно из ноутбука. Он также использует модуль pypki2 за кулисами для обработки HTTPS-соединений, которым требуются ваши сертификаты PKI.

```
импорт ipydeps
ipydeps.pip( 'prettytable' )
```

(U // FOUO) Еще одна вещь, которую ipydeps делает за кулисами, - это пытается установить зависимости операционной системы (не связанные с python), которые необходимы пакету для правильной установки и запуска. Это настраивается вручную командой Jupytfer здесь, в АНБ. Если у вас возникнут проблемы с установкой пакета с ipydeps в Jupyter на LABBENCH, свяжитесь с [УДАЛЕНО] и укажите название пакета, который вы пытаетесь установить, и ошибки, которые вы видите.

Модули и пакеты

Обновлено почти 3 года назад [УДАЛЕНО] в COMP 3321 (U) Урок 08: Модули и пакеты

(U) Я вижу, тебе нравится Python, поэтому я вставил Python в твой Python

(U) Мы видели, как писать скрипты; теперь мы хотим повторно использовать хорошие части. Мы уже использовали команду import, которая позволяет нам подключаться к работе других пользователей - либо с помощью обширной стандартной библиотеки Python, либо с помощью дополнительных, отдельно устанавливаемых пакетов. Это также может быть использовано для активного использования длинного хвоста нашего собственного производства. В этом уроке мы гораздо подробнее рассмотрим механику и принципы написания и распространения модулей и пакетов.
Предположим, у вас есть скрипт с именем my_funcs.py в вашем текущем каталоге. Тогда следующее работает просто отлично:

```
импорт my_functs
импорт my_functs как m

импорт importlib
importlib.перезагрузить (m)

из my_funcs импортировать string_appender
из my_functs импорт * # БУДЬ ОСТОРОЖЕН
```

(U) Если вы измените исходный файл my_funcs.py в промежутках между командами импорта у вас будут импортированы разные версии функций. Так что же происходит?

(U) Пространства имен

(U) Когда вы импортируете модуль (то, что мы привыкли называть просто скриптом), Python выполняет его как бы из командной строки, затем помещает переменные и функции внутри пространства имен, определенного именем script (или используя необязательное ключевое слово as). Когда вы из <module> импортируете <name>, переменные импортируются в ваше текущее пространство имен. Представьте себе пространство имен как суперпеременную, которая содержит ссылки на множество других переменных, или как суперкласс, который может содержать данные, функции и классы. (U) После импорта модуль становится динамичным, как и любой объект Python; например, функция reload принимает модуль в качестве аргумента, и вы можете добавлять данные и методы в модуль после того, как вы его импортировали (но они не будут сохраняться в течение срока службы вашего скрипта или сеанса).

```
импортировать my_functs как m

def silly_func (x) :
    возвращение "Глупый {}!" .формат(x)

m.silly_func = silly_func
m.silly_func( "Метка" )
# Глупый Марк!
```

(U) Напротив, команда from <модуль> import <функция> добавляет функцию в текущее пространство имен.

(U) Предотвращение избыточной производительности: магия main

(U) Предположим, у вас есть скрипт, который делает что-то потрясающее, называется awesome.py :

```
класс Потрясающий(объект):
    защита __инициализация__(самооценка, awesome_thing):
        self.thing = awesome_thing
    def __str__(самостоятельно):
        Возврат "{0.вещь} ПОТРЯСАЮЩАЯ".format (self)

a = Потрясающе( "СБРОС БАЗЫ" )
вывести(а)
```

(U) Это может быть выполнено из командной строки или принудительно:

```
(VENV) [УДАЛЕНО] $ python awesome.py
Бейсджампинг - это КРУТО
(VENV) [УДАЛЕНО] $ python
импортировать awesome
бейсджампинг - это КРУТО.

а

Обратная трассировка.последний вызов):
Файл "<stdin>", строка 1, в <модуль>
Ошибка имени: имя 'a' не определено
awesome.a
awesome.Потрясающий объект в 0x7fa222a8b410>
печать (потрясающе.а)
Бейсджампинг - это ПОТРЯСАЮЩЕ.
```

(U) Вы же не хотите, чтобы эта инструкция print выполнялась каждый раз, когда вы ее импортируете. Не менее важно, awesome.a, вероятно, является посторонним элементом при импорте. Давайте исправим это, чтобы избавиться от них при импорте модуля, но сохранить их при выполнении скрипта.

```
класс Потрясающий(объект):
    определение __инициализация__(самооценка, awesome_thing):
        self.thing = awesome_thing
    def __str__(самостоятельно):
        Возврат "{0.thing} ПОТРЯСАЮЩЕ".format (self)
    def main ():
        a = Потрясающий( "БАЗОВЫЙ сброс" )
        вывести(а)
```

```
если  __name__ ==  '__main__'  :
    a = Потрясающе(  "Бейсджампинг"  )
    вывести(a)
```

(U) Мы можем сделать еще лучше. Есть некоторые ситуации, например, профилирование или тестирование, когда мы хотели бы импортировать модуль, а затем посмотреть, что произойдет, если мы запустим его как скрипт. Чтобы включить это, переместите основную функциональность в функцию с именем main() :

```
класс  Потрясающий(объект):
    определение инициализация(самооценка, awesome_thing):
        self.thing = awesome_thing
    def  __str__  (самостоятельно):
        Возврат  "{0.thing} ПОТРЯСАЮЩЕ".  .формат (self)

def  main  ():
    a = Потрясающий(  "Бейсджампинг"  )
    вывести(a)

если  __name__ ==  '__main__'  :
    main()
```

(U) От модулей к пакетам

(U) Файлу соответствует отдельный модуль Python. Нетрудно представить ситуацию, когда у вас есть несколько связанных модулей, которые вы хотите сгруппировать вместе в эквивалент папки; термин Python для обозначения этой концепции - package . Мы изготавливаем упаковку с помощью

создаем папку,

помещаем в нее скрипты / модули

добавляем некоторую магию Python (которая, очевидно, будет каким-то образом включать __ shape или форму) (U) Например, мы поместим awesome.py в пакете под названием feelings-позже мы добавим terrible.py и totally_rad.py Структура каталогов такова:

```
чувства/ |— awesome.py |— init.py |— main.py
```

(U) Требуется файл init.py; без него Python не идентифицирует эту папку как пакет. Однако, main.py является необязательным, но приятным; если он у вас есть, вы можете ввести python feelings, и содержимое main.py будет выполнено в виде скрипта. (Примечание: Теперь вы можете постулировать, что

```
если  __name__ ==  '__main__'  :
```

действительно работает. (U) Файл init.py может содержать команды. Очень похоже на функцию init() класса init.py выполняется сразу после импорта пакета. Одно из распространенных применений - разоблачать

модули как атрибуты пакета; для этого требуется всего лишь импортировать.имя_модуля> в файл пакета init.py.

(U) Вперед, ко всему миру

Довольно скоро вам захочется поделиться пакетами feelings с более широкой аудиторией. Есть тысячи людей, которые хотят делать потрясающие вещи, но у них нет времени создавать свою собственную версию, которая в любом случае будет не так хороша, как ваша, поэтому они рассчитывают, что вы предоставите этот пакет в. удобном и простом в установке виде.

(U) Пакеты, доступные для совместного использования

(U) Пакет _setuptools (который построен на distutils), используемый совместно с виртуальными средами и общедоступными репозиториями в системах контроля версий, делает совместное использование вашей работы таким же простым, как установка pip. пакет от PyPI. Вы используете систему контроля версий, не так ли ? В этом уроке предполагается, что вы используете git и отправляете свои репозитории в GitLab (U) Чтобы сделать пакет feelings доступным для всего мира, его следует поместить в корневой каталог репозитория git, рядом со скриптом установки под названием setup.py , т.е.

feelings_repo |— чувства/ |— awesome.py |— init.py |— main.py |— setup.py

(U) setup.py Скрипт выполняет импорт из одного из двух пакетов, которые обрабатывают управление и установку других пакетов. В этом примере мы будем использовать setuptools, потому что он более мощный и устанавливается по умолчанию в виртуальных средах. В простых случаях, подобных этому, встроенный модуль distutils более чем адекватен и функционально идентичен. (U) Скрипт вызывает. единственная функция, setup, и принимает метаданные о пакете, включая название и номер версии пакета, имя и адрес электронной почты разработчика, ответственного за пакет, и. список пакетов.или модулей). Выглядит это так:

```
из setuptools импорт настройка
setup(name= "pyTest" ,
    версия= '0.0.1' ,
    описание= "Самый простой пакет Python, который только можно себе представить" ,
    автор= "[УДАЛЕНО]" ,
    адрес электронной почты автора= "[УДАЛЕНО]" ,
    посылки=[ "чувства" ],
)
```

(U) Чтобы использовать distutils вместо setuptools, измените первую строку на чтение из distutils.core import setup. Двумя мощными преимуществами setuptools перед distutils являются:

- Управление зависимостями, так что внешние пакеты, доступные в PyPI, будут установлены автоматически, и
- автоматическое создание сценариев оболочки точки входа, которые подключаются к указанным функциям в вашем коде.

(U) Совместное использование пакетов

(U) У нас есть дела поважнее - мы хотим донести Потрясающее до всего мира, и мы почти у цели. Как только изменения будут зафиксированы и отправлены в GitLab, мы сможем поделиться ими с помощью одной простой команды pip.

Выполнить внутри виртуальной среды может любой пользователь, имеющий доступ к GitLab

```
$ pip install -e git+git@gitlab.coi.nsa.ic.gov:[УДАЛЕНО]/feelings.git #egg=чувства
```

(U) Флаг -e устанавливает репозиторий как доступный для редактирования, также в режиме разработчика. Это означает, что полный репозиторий git клонируется внутри папки src виртуальной среды и может быть изменен или обновлен на месте. например, с помощью git clone) без необходимости переустановки. Параметр #egg=feelings необходим для работы установки pip и должен быть добавлен вручную; он не требуется и даже не используется GitLab. (U) Как только ваш пользователь установит ваш пакет pip, все! Теперь она может делать потрясающие вещи, например,

```
из чувств import awesome

a = awesome. Потрясающе( "Достоевский" )

вывести(а)

# Достоевский ПОТРЯСАЮЩИЙ.
```

(U) Еще лучше то, что ей потребуется совсем немного больше усилий, чтобы включить ваш пакет в качестве. зависимость в ее пакетах и приложениях!

Урок 09: Исключения, профилирование и тестирование

Обновлено 8 месяцев назад [УДАЛЕНО] в COMP 3321 (U) Обработка исключений, тестирование кода и профилирование в Python.

НЕКЛАССИФИЦИРОВАНО

(U) Введение

(U) Внимание к обработке исключений, профилированию и тестированию отличает профессиональных разработчиков, пишущих высококачественный код, от любителей, которые халтурят только для того, чтобы выполнить работу. Каждая тема требует многочасового обсуждения сама по себе, но Python позволяет начать изучать и использовать эти принципы с минимальными усилиями. В данном разделе описываются основные идеи заинтересовать читателей и увидеть полезность этих идей и модулей. Давайте begin...by делая некоторые ошибки.

(U) Исключения

(U) Python очень гибкий и сделает все возможное, чтобы выполнить все, о чем вы его попросите, но иногда вы можете просто слишком сильно запутать его. Первый тип ошибок - это синтаксические ошибки. К этому моменту в курсе мы все видели их более чем достаточно! Они возникают, когда Python не может разобрать то, что вы ввели.

```
для меня в диапазоне (10):
    def altered_cool () :
        печать(потрясающая.Потрясающая( "Домашний уксус" )) # все еще там?
        печать(coolgroup( "интеллигенция" ))
        круто = "хипстер"
        лесоруб = Верно
        распечатать(отсортировано(locals().keys()))
        печать(местные жители()[ "круто" ])

altered_cool()

'дровосек' в globals()

globals() [ "круто" ]

глобальные() == локальные()
```

(U) Пространства имен модулей

(U) Наконец, у каждого модуля также есть свое собственное пространство имен модулей. Вы можете проверить их, используя специальный метод dict модуля.

```
отсортировано(awesome.__dict__.keys())
# угадай что?
dir(awesome) == отсортировано(awesome.__dict__. ключи())

потрясающе.__dict__ [ "круто" ] # может ли aiso распечатать это, чтобы получить ячейку памяти

# разве мы только что не видели этого здесь?
globals ()[ 'coolgroup' ]

режиссер (awe)

awe.__диета__[ "круто" ]

awe -= потрясающий

идентификатор (awe) == идентификатор (awesome)
```

(U) Изменение пространств имен модулей

(U) Вы можете добавлять в пространства имен модулей "на лету". Однако имейте в виду, что это будет длиться только до завершения работы программы, а фактический файл модуля останется неизменным.

```
определение более привлекательно () :
    возврат "Они потрясающие!"

awe.exclaim = еще__awesome

awe.exclaim()

"воскликни" в реж ("благоговейный трепет")

"воскликни" в реж (потрясающе)
```

(U) Пакеты

(U) Что делать, если вы хотите разбить свой код на несколько модулей? Поскольку модуль представляет собой файл, естественно собрать все связанные модули в один каталог или папку. И, действительно, пакет Python - это всего лишь каталог, содержащий модули, специальный файл init.py и иногда больше пакетов или других вспомогательных файлов.

```
Файл "<ipython-input-l-6f7914dd2e9a>" , строка 1
для я в диапазон(10)
                ^
SyntaxError: неверный синтаксис
```

(U) Python не смог разобрать то, что мы пытались здесь сделать (потому что мы забыли наше двоеточие). Это, однако, дало нам знать, где все перестало иметь смысл. Обратите внимание на напечатанную строку с крошечной стрелкой (^), указывающей на то, где, по мнению Python, есть проблема. (U) Оператор SyntaxError: недопустимый синтаксис является примером специального исключения, называемого SyntaxError. Довольно легко увидеть, что произошло здесь, и делать особо нечего, кроме как исправить вашу опечатку. Другие исключения могут быть гораздо более интересными. (U) Существует много типов исключений:

```
импорт встроенных
# Эта мельница показывает большую производительность.
# Чтобы сделать ее прокручиваемой, выделите эту ячейку и выберите
# Ячейка> Текущий вывод> Переключить прокрутку
справка (встроенная)

# Python 2 раньше содержал эту информацию в модуле "исключения"
# Python 3 переместил ее во "встроенные" для обеспечения согласованности
# Итак, для python 2 попробуйте вместо этого следующее:
```

```
импорт исключений
каталог (исключения)
```

(U) Бьюсь об заклад, мы сможем воплотить некоторые из них в жизнь. На самом деле, вы, вероятно, уже сделали это недавно.

```
1/0

защита f ():
    1/0

f()
1/ '0'

импорт крис

файл = открыть( 'data' , 'w' )

file.read()
```

(U) Обработка исключений

(U) Когда могут возникать исключения, лучший способ справиться с ними и сделать что-то более полезное, чем выйти и вывести что-либо на экран. На самом деле, иногда исключениями могут быть очень полезные инструменты.например, Keyboardlnterrupt). В Python мы обрабатываем исключения с помощью команд try и except . (U) Вот как это работает:

1. (U) Выполняется все, что находится между командами try и except.
2. (U) Если это не приводит к возникновению исключения, блок except пропускается и программа продолжается.
3. (U) Если возникает исключение, остальная часть блока try пропускается.
4. (U) Если тип исключения назван после ключевого слова except, выполняется код после команды except .
5. (U) В противном случае выполнение останавливается, и у вас возникает необработанное исключение. (U) Все имеет больше смысла на примере:

```
def .(x):
    попробуйте
        распечатать("Я собираюсь преобразовать входные данные в целое число" )
        печать(int(x))
    за исключением ValueError:
        печать( "Извините, я не смог преобразовать это". )
f(2)
f( '2' )
f( "два" )
```

(U) В команду except можно добавить несколько типов исключений:

```
за исключением(Ошибка типа, ошибка значения):
```

(U) Ключевое слово as позволяет нам извлечь сообщение из ошибки:

```
def be_careful (a, b):
    попробовать
        распечатать((float.a)/(float.b))
    исключение (ValueError, TypeError, ZeroDivisionError) как подробно:
        печать( "Обработанное исключение": , подробно)
    кроме :
        распечатать("Непредвиденная ошибка!" )
    наконец :
        распечатать("ЭТО БУДЕТ ВЫПОЛНЯТЬСЯ ВСЕГДА!")
будь осторожен(1,0)
будь осторожен(1, [1,2])
будь осторожен(1, "два" )
be_c. refu1(16** 400,1)
float(16**400)
```

(U) Мы также добавили команду finally . Он всегда будет выполняться, независимо от того, было исключение или нет, поэтому его следует использовать как место для очистки всего, что осталось от предложений try и except , например закрытия файлов, которые все еще могут быть открыты.

(U) Создание исключений

(U) Иногда вам захочется вызвать исключение и позволить кому-то другому обработать его. Это можно сделать с помощью команды raise.

```
поднять Ошибка типа( "Вы указали неправильный тип" )
```

(U) Если ни одно встроенное исключение не подходит для того, что вы хотите создать, определите. новый тип исключения так же прост, как и создание. новый класс, наследующий от типа исключения.

```
класс MyPersonalError (Исключение) :
    пас
повышение MyPersonalError( "Я могущественен. Услышь мой рев!" )

определение локатор (myLocation):
    если (myLocation<0):
        повысить уровень MyPersonalError( "Я могуч. Услышь мой рев!" )
    печать (myLocation)
определитель местоположения(-1)
```

(U) При перехвате исключения и его возбуждении. другой, оба исключения будут raised.as из Python ..3).

```
класс MyException (Исключение):
    пройти

попробовать
    int( "abc" )
за исключением ValueError:
    увеличить MyException( "Вы не можете преобразовать текст в целое число!" )
```

(U) Вы можете переопределить это, добавив синтаксис из None в конец вашего оператора raise.

```
класс MyException ( Исключение ):
    пройти
попробовать
    int( "abc" )
за исключением ValueError:
    увеличить MyException( "Вы не можете преобразовать текст в целое число!" ) От Нет
```

(U) Тестирование

(U) Есть два встроенных модуля, которые довольно полезны для тестирования вашего кода. Это также позволяет тестировать код каждый раз, когда он импортируется, чтобы пользователь на другом компьютере заметил, если определенные методы не сделали того, для чего они были предназначены заранее.

(U) Модуль doctest

(U) Модуль doctest позволяет тестировать код и утверждения значений в документации к самому коду. Это также работает с исключениями; вы просто копируете и вставляете соответствующую обратную трассировку , которая ожидается (необходимы только первая строка и фактическая строка исключения). Вы можете включить doctest в модуль или скрипт. Подробности смотрите в официальной документации Python.

```
"""
Это модуль "пример".

Пример модуля предоставляет одну функцию, factorial(). Например,

>>>     факториал(5)
120
"""
def     факториал       (n) :
    """Возвращает факториал., точное целое число >= 0.
    >>> [факториал (n) для n в диапазоне (6)]
    [1, 1, 2, 6, 24, 120]

    >>> факториал(30)
    265252859812191058636308480000000

    >>> факториал(-1)
    Обратная трассировка (последний последний вызов):
    Ошибка значения: n должно быть >= 0

    Факториалы с плавающей точкой в порядке, но значение с плавающей точкой должно быть точным целым числом:
    >>> факториал(30.1)
    Обратная трассировка (последний последний вызов):
    ValueError: n должно быть точным целым числом

    >>> факториал(30.0)
    265252859812191058636308480000000

    Он также не должен быть смехотворно большим:
    >>> факториал(lel00)
    Обратная трассировка (последний последний вызов):
    OverflowError: n слишком велико
    """

    импорт      математика
    если нет    n>= 0:
        повысить   ValueError (          "n должно быть >= 0"          )
    if      math.floor(n) != n:
        повысить   ValueError(          "n должно быть точным целым числом"          )
    если  n+1 == n:          # получить значение, подобное 1e300
        повысить   OverflowError(          "n слишком велико"      )
    результат = 1
    коэффициент = 2
    в то время как фактор <= n:
        результат *= фактор
        фактор += 1
    возвращает   результат

если  _name_ ==          " __main__ "          :
    импорт      doctest
    doctest.testmod()
```

(U) Этот урок может быть сложным для понимания из записной книжки. Он будет наиболее осмысленным, если вы скопируете и вставите приведенный выше код в. файл с именем factorial.py, затем из терминала запустите:

```
факториал python .py -v
```

Обратите внимание, что вам не обязательно включать строки doctest в свой код. Если вы удалите их, должно сработать следующее:

```
python -m doctest -v факториал .py
```

(U) Модуль unittest

(U) Модуль unittest гораздо более структурирован, что позволяет разработчику создавать. класс тестов, которые запускаются и анализируются гибко. Чтобы создать модульный тест для модуля или скрипта:

импортируйте unittest,

создайте тестовый класс как подкласс unittest.Введите TestCase,

добавьте тесты в качестве методов этого класса, убедившись, что имя каждой тестовой функции начинается со слова "test", и

добавьте unittest.main() в ваш основной цикл для запуска тестов.

```
импорт единичный
# ... другой импорт, код скрипта и т.д. ...
класс Факторные тесты (единичный тест.TestCase):
    определение testSinglevalue (self):
        self.assertEqual(факториал(5), 120)

    определение неоднозначные тесты (self):
        self.assertRaises(ошибка типа, факториал, [1,2,3,4])

    def testBoolean (самостоятельно):
        self.assertTrue(факториал(5) == 120)

def main () :
    """ Основная функция для этого скрипта """
    unittest.main() # Проверьте документацию на наличие дополнительных уровней детализации и т.д.
    # ... остальная часть основной функции ...

если _name_ == "__главный___" :
    main()

импорт unittest
каталог (unittest.ТестОвый кейс)
```

(U) Профилирование

(U) Существует множество модулей профилирования, но мы продемонстрируем модуль cProfile из стандартной библиотеки. Чтобы использовать его в интерактивном режиме, сначала импортируйте модуль, затем вызовите его с помощью. единственный аргумент, который должен быть. строка, которая могла бы быть выполнена, если бы она была введена в интерпретатор. Часто это будет ранее определенная функция.

```
импорт cProfile
def long (верхний предел=100000):
    для x в диапазоне (верхний предел):
        пас

защита короткий ():
        проход

защита внешний (верхний предел=100000):
        короткий()
        короткий()
        длинный()

cProfile.run( 'outer()' )

cProfile.run( 'внешний(10000000)' )
```

(U) На выходе отображается

```
ncalls  : количество вызовов,
общее время : общее время, проведенное в данной функции (и исключая время, затраченное на вызовы подфункций),
количество звонков: частное от  общего времени, деленное на  количество звонков
общее время : общее время, затраченное на выполнение этой и всех подфункций (от вызова до выхода). Это точное значение
percall: частное от  времени обработки, разделенное на вызовы примитивов
имя файла:lineno(  функция): предоставляет соответствующие данные для каждой функции
```

(U) Быстрый и простой способ профилирования. все приложение состоит просто из вызова основной функции cProfile с вашим скриптом в качестве дополнительного аргумента:

```
cProfile $ python.m myscript.py
```

(U) Еще один полезный встроенный профилировщик - timeit. Он хорошо подходит для быстрых ответов на такие вопросы, как "Что лучше между A и B?"

```
$ python -m timeit  "'для. в диапазоне(100):' ' str(i)'
```

```
импорт  timeit
timeit.timeit(  "'-".присоединиться(str(n) для. в диапазоне(100))'  ,число=20000)
mySetup =  '''
определение myfunc(верхний предел=100000):
возвращает диапазон (верхний предел)
'''
timeit.timeit(  'myfunc()'  , число=1000, настройка= mySetup)
```

Упражнение 1. Напишите пользовательскую ошибку и вызовите ее, если RangeQuery создан с датами не в правильном формате.

Упражнение 2: Дан список кортежей: [("2016-12-01", "2016-12-06"),("2015-12-01", "2015-12-06"), ("2016-2-01", "2016-2-06"),("01/03/2014", "02/03/2014"), ("2016-06-01", "2016-10-06")] напишите цикл для печати запроса диапазона для каждого из диапазонов дат, используя "TS / / SI // REL TO USA, FVEY", "Основной IP-адрес Zendian diplomat", "10.254.18.162" в качестве классификации, обоснования и селектора.

Внутри цикла напишите. try/except block, чтобы перехватить вашу пользовательскую ошибку из-за неправильно отформатированных дат.

НЕСЕКРЕТНО

Урок 10: Итераторы, генераторы и "Утиный"

ввод

Обновлено 9 месяцев назад [УДАЛЕНО] в COMP 3321 (U) Итераторы, генераторы, сортировка и утиный ввод в Python.

НЕСЕКРЕТНО

(U) Введение: пересмотренные представления о перечне

(U) Мы начнем с рассмотрения основ составления списков и понимания их содержания.

```
melist = [ я для я в диапазоне (1, 100, 2) ]
для я в melist: # /low работает ли цикл?
    печать(i)
```

(U) Что происходит, когда построение списка усложняется?\

```
noprimes = [ j для i в диапазоне (2, 19) для j в диапазоне (i*2, 500, i) ]
простые числа = [ x для x в диапазоне (2 500) если x не в noprimes ]
печать(отсортированных(простых чисел))
```

(U) Можем ли мы сделать это одним выстрелом? Да, но...

```
# гнездящееся безумие!
простые числа = [ x для x в диапазоне (2, 500) если x не в [j для i в диапазоне (2, 19) для j в диапазоне (i*2,
```

(U) Итераторы

(U) Чтобы создавать свои собственные итеративные объекты, пригодные для использования в циклах и для понимания списков, все, что, ,, вам нужно сделать, это реализовать правильные специальные методы для класса. Метод iter должен возвращать сам итерируемый объект (почти всегда self), а следующий метод определяет значения итератора. (U) Давайте приведем пример, придерживаясь ранее представленной темы, итератора, который возвращает числа по порядку, за исключением кратных аргументов, использованных во время построения. Мы убедимся, что это в конечном итоге завершается, вызывая исключение stopiteration всякий раз, когда оно достигает 200. (Это отличный пример исключения в Python, которое не является чем-то необычным: обработка события, которое не является неожиданным, но требует завершения; для циклов и понимания списка ожидайте получить

исключение StopIteration как сигнал к остановке обработки.)

```
класс Не поддающийся фильтрации фактор объект ) :
    def __init__ (self, *аргументы):
        self.avoidjnultiples = аргументы
        self.x = 0
    определить следующий (self):
        самость.x += 1
        пока Верно :
            если селф.x> 200:
                поднять Стопитерация
            для y в self.avoid_multiples:
                если self.x % y == 0:
                    самость.x += 1
                    перерыв
            else :
                return self.x

    def __iter__ (self):
        возврат self

silent_fizz_buzz = нечитаемый (3, 5)

[x для x в silent_fizz_buzz]

mostly_prime = нечитаемый(2, 3, 5, 7, 11, 13, 17, 19)

частичная сумма = 0

для x в в основном простое:
    частичная сумма += x

частичная сумма

mostly_prime = без учета факторов(2, 3, 5, 7, 11, 13, 17, 19)
вывести (сумма (mostly_prime))
```

(U) Может показаться странным, что метод iter, похоже, ничего не делает. Это связано с тем, что в некоторых случаях итератор для объекта не должен совпадать с самим объектом. Рассмотрение такого использования выходит за рамки курса. (U) Существует другой способ реализации. пользовательский итератор: метод getitem. Это позволяет вам использовать обозначение в квадратных скобках [] для получения данных из объекта. Однако вы все равно должны помнить о повышении. исключение stopiteration для его правильной работы в циклах for и понимании списков.

Другой пример итератора

В приведенном ниже примере мы создаем итератор, который возвращает квадраты чисел. Обратите внимание, что в

следующий метод, все, что мы делаем, это повторяем наш счетчик (self.x) и возвращаем квадрат этого номера счетчика, при условии, что счетчик не превышает заранее определенного предела (self.limit).
Цикл while в предыдущем примере был специфичен для этого варианта использования; на самом деле нам вообще не нужно реализовывать какой-либо цикл. далее, поскольку это просто метод, вызываемый для каждой итерации через цикл в нашем итераторе. Здесь мы также реализуем . getitem . метод, который позволяет нам извлекать. значение из итератора в. определенном местоположении индекса. Этот просто вызывает итератор с помощью self.next, пока он не достигнет желаемого местоположения индекса, затем возвращает это значение.

```
класс Квадраты (объект):
    определение __инициализация__(self, limit=200):
        self.limit = ограничение
        self.x = 0

    определение __следующий__(самостоятельно):
        self.x += 1
        если self.x > self.limit:
            поднять Остановка
        возврат (self.x-1)**2

    определение __getitem__ (self, idx):
        # инициализировать счетчик на.
        self.x = 0
        если нет isinstance(idx, int):
            вызвать Исключение ( "Принимаются только аргументы с целочисленным индексом!" )
        в то время как self.x < idx:
            ясам.__следующий__()
        возврат к себе.x**2

        определение __iter__ (self):
            вернуть себя

my_squares = Квадраты (ограничение= 20)

[x для x в my_squares]

my_squares[5]

# поскольку мы установили ограничение в 20, мы не можем получить доступ к местоположению индекса выше этого
my_squares[25]
```

(U) Преимущества пользовательских итераторов

1. (U) Более чистый код
2. (U) Возможность работать с бесконечными последовательностями
3. (U) Возможность использовать встроенные функции, такие как sum, которые работают с итерациями
4. (U) Возможность сохранения памяти (например, диапазона)

(U) Генераторы

(U) Generators are iterators with a much lighter syntax. Very simple generators look just like list comprehensions, except they're surrounded with parentheses () instead of square brackets []. More complicated generators are defined like functions, with the one difference being that they use the yield keyword instead of the return keyword. A generator maintains state in between times when it is called; execution resumes starting immediately after the yield statement and continues until the next yield is encountered.

```
y = (x*x for x in range(30))
print(y) # hmm …

def xsquared():
    for i in range(30):
        yield i*i

def xsquared_inf():
    x = 0
    while True:
        yield x*x
        x += 1

squares = [x for x in xsquared()]
print(squares)
```

(U) Another example…days of the week!

```
def day_of_week():
    i = 0
    days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
    while True:
        yield days[i%7]
        i += 1
day_of_week()

import random
def snowday(prob=.01):
    r = random.random()
    if r < prob:
        return "snowday!"
    else:
        return "regular day."
n = 0
for x in day_of_week():
    today = snowday()
    print(x + " is a " + today)
    n += 1
```

```
    if today == "snowday!" :
        break

weekday = (day for day in day_of_week())
next.weekday()
```

(U) Pipelining

(U) One powerful use of generators is to connect them together into a pipeline, where each generator is used by the next. Since Python evaluates generators "lazily," i.e. as needed, this can increase the speed and potentially allow steps to run concurrently. This is especially useful if one or two steps can take a long time (e.g. a database query). Without generators, the long-running steps will become a bottleneck for execution, but generators allow other steps to proceed while waiting for the long-running steps to finish.

```
import random
# Get the fractional, part of. string representation of. float
def frac_part (v) :
    v = str(v)
    i, f = v.split( '.' )
    return f

# traditional approach
results = []
for i in range(20):
    r = random.random() *100 # generate a random number
    r_str = str(r) # convert it to a string
    r_frac = frac_part(r_str) # get the fractional part
    r_out = float( '0.' + r_frac) # convert it back to a float
    results.append(r_out)

results

# generator pipeline
rand_gen = ( random.random() * 100 for i in range(20) )
str_gen = ( str(r) for r in rand_gen )
frac_gen = ( frac_part(r) for r in str_gen )
out_gen = ( float( '0.' +r) for r in frac_gen )

results = list(out_gen)
results
```

(U) Sorting

(U) In Python 2, anything iterable can be sorted, and Python will happily sort it for you, even if the data is of mixed types--by default, it uses the built-in cmp function, which almost always does

something (except with complex numbers). However, the results may not be what you expect! (U) In Python 3, iterable objects must have the lt (lt = less than) function explicitly defined in order to be sortable. (U) The built-in function sorted(x) returns a new list with the data from x in sorted order. The sort method (for lists only) sorts. list in-place and returns None .

```
int_data = [10, 1, 5, 4, 2]
sorted(int_data)
int_data
int_data.sort()
int_data
```

(U) To specify how the sorting takes place, both sorted and sort take an optional argument called key. key specifies a function of one argument that is used to extract a comparison key from each list element (e.g. key=str.lower). The default value is None (compare the elements directly).

```
users = [ 'hAckerl' , 'TheBoss' , 'botman' , 'turingTest' ]
sorted(users)
sorted(users, key=str.lower)
```

(U)The it function takes two arguments: self and another object, normally of the same type.

```
class comparableCmp (complex):
    def __lt__ (self, other):
        return abs(self) < abs(other)

a = 3+4j
b = 5+12j
a < b
a1 = comparableCmp(a)
b1 = comparableCmp(b)
a1 < b1
c = [b1, a1]
sorted(c)
```

(U) Here's how it works:

1. the argument given to key must be a function that takes a single argument;
2. internally, sorted creates function calls key(item) on each item in the list and then
3. sorts the original list by using It on the results of the key(item) function. (U) Another way to do the comparison is to use key :

```
def magnitude_key (a) :
    return (a*a.conjugate()).real
```

```
magnitude_key(3+4j)

sorted([5+3j, 1j, -2j, 35+0j], key=magnitude_key)
```

(U) In many cases, we must sort. list of dictionaries, lists, or even objects. We could define our own key function or even several key functions for different sorting methods:

```
list_to_sort = [{ 'lname' : 'Dones' , 'fname' : 'Sally' },
                { 'lname' : 'Dones' , 'fname' : 'Derry' },
                { 'lname' : 'Smith' , 'frame' : 'Dohn' }]

def lname_sorter (list_item):
    return list_item[ 'lname' ]
def fname_sorter (list_item):
    return list_item[ 'fname' ]
def lname_then_fname_sorter (list_item) :
    return (list_item[ 'lname' ], list_item[ 'fname' ])

sorted(list_to_sort, key=lname_sorter)
sorted(list_to_sort, key=fname_sorter)
sorted(list_to_sort, key=lname_then_fname_sorter)
```

(U) While it's good to know how this works, this pattern common enough that there is. method in the standard library operator package to do it even more concisely.

```
import operator
lname_sorter = operator.itemgetter( 'lname' ) # same as previous lname_sorter
```

(U) The application of the itemgetter method returns. function that is eqivalent to the lname_sorter function above. Even better, when passed multiple arguments, it returns. tuple containing those items in the given order. Moreover, we don't even need to give it. name first, it's fine to do this:

```
sorted(list_to_sort, key=operator.itemgetter( 'lname' ))

sorted(list_to_sort, key=operator.itemgetter( 'lname' , 'fname' )) # same as using lname_then_fname_sorter
```

(U) To use operator, itemgetter with lists or tuples, give it integer indices as arguments. The equivalent function for objects is operator.attrgetter (U) Since we know so much about Python now, it's not hard to figure out how simple operator.itemgetter actually is; the following function is essentially equivalent:

```
def itemgetter_clone (*args) :
    def f (item):
```

```
        return tuple(item[x] for x in args)
    return f
```

(U) Obviously, operator.itemgetter and itemgetter_clone are not actually simple-it's just that most of the complexity is hidden inside the Python internals and arises out of the fundamental data model.

(U) Duck Typing

(U) All the magic methods we've discussed are examples of the fundamental Python principle of duck typing: "If it walks like a duck and quacks like a duck, it must be a duck." Even though Python has isinstance and type methods, it's considered poor form to use them to validate input inside a function or method. If verification needs to take place, it should be restricted to verifying required behavior using hasattr . The benefit of this approach can be seen in the built-in sum function.

```
help(sum)
```

(U) Any sequence of numbers, regardless of whether it's. list, tuple, set, generator, or custom iterable, can be passed to sum. (U) The following is. comparison of bad and good examples of how to write a product function:

```
def list_prod (to_multiply):
    if isinstance(to_multiply, list): # don't do this!
        accumulator = 1
        for i in to_multiply:
            accumulator *= i
        return accumulator
    else :
        raise TypeError( "Argument to_multiply must be a list" )


def generic_prod (to_multiply):
    if hasattr(to_multiply, '__iter__' ) or hasattr(to_multiply, '__getitem__' ):
        accumulator = 1
        for i in tojnultiply:
            accumulator *= i
        return accumulator
    else :
        raise TypeError( "Argument to_multiply must be a sequence" )

list_prod([1,2,3])
list_prod((1,2,3))
generic_prod((1,2,3))
```

(U) Having given that example, testing for iterability is one of a few special cases where isinstance might be the right function to use, but not in the obvious way. The collections package provides abstract base classes which have the express purpose of helping to determine when an object implements. common interface.

(U) Finally, effective use of duck typing goes hand in hand with robust error handling, based on the principle that "it's easier to ask for forgiveness than permission."

Exercises

1. Add a method to your 'RangedQuery' class to allow instances of the class to be sorted by 'start_date'.
2. Write an iterator class 'Reverselter' that takes a list and iterates it from the reverse direction.
3. Write a generator which will iterate over every day in a year. For example, the first output would be 'Monday, January 1'.
4. Modify the generator from exercise 2 so the user can specify the year and initial day of the week.

UNCLASSIFIED

Pipelining with Generators

Created over 3 years ago by [DELETED] (U) Defining processing pipelines with generators in Python. It's simply awesome.

Pipelining with Generators

Imagine you're doing your laundry. Think about the stages involved. Roughly speaking, the stages are sorting, washing, drying, and folding. The beauty though is that even though these stages are sequential, they can be performed in parallel. This is called pipelining.

Python generators make pipelining easy and can even clarify your code quite a bit. By breaking your processing into distinct stages, the Python interpreter can make better use of your computer's resources, and even break the stages out into separate threads behind the scenes. Memory is also conserved because values are automatically generated as needed, and discarded as soon as possible.

A prime example of this is processing results from. database query. Often, before we can use the results of a database query, we need to clean them up by running them through. series of changes or transformations. Pipelined generators are perfect for this.

```
from pprint import pprint
import random
```

A Silly Example

Here we're going to take 200 randomly generated numbers and extract their fractional parts (the part after the decimal point). There are probably more efficient ways to do this, but we're doing to do it by splitting out the string into two parts. Here we have a function that simply returns the integer part and the fractional part of an input float as two strings in a tuple.

```
def split_float (v) :
    """
    Takes a float or string of a float
    and returns a tuple containing the
    integer part and the fractional part
    of the number, as strings, respectively.
    """
    v = str(v)
    i, f = v.split( '.' )
    return (i, '0.' +f)
```

The Pipeline

Here we have a pipeline of four generators, each feeding the one below it. We pprint out the final resulting list after all the stages have complete. See the comments after each line for further explanation.

```
rand__gen = (random.random() * 100 for i in range(200) ) # generate 200 random fLoots between. and 100, one at
results = (split_float(r) for r in rand_gen ) # call our spiit_fLoat() function which will generate the corres
results = (r[l] for r in results ) # we onLy care about the fractionai part, so only beep that part of the tup
results = (float(r) for r in results ) # convert our fractional value from a string bach into a float
pprint(list(results)) # print the final results
```

Why not a for-loop?

We could have put all the steps of our pipeline into a single for-loop, but we get a couple advantages by breaking the stages out into separate generators:

There's some clarity gained by having distinct stages specified as a pipeline. People reading the code can clearly see the transforms.

In a for-loop, Python simply computes the values sequentially; there's no chance for automatic

optimization or multi-threading. By breaking the stages out, each stage can execute in parallel, just like your washer and dryer.

Another(Pseudo-)Example

Here's a pseudo-example querying. database that returns JSON that we need to convert to lists.

```
import json
results = ( json.loads(result) for result in db_cursor.execute(my_query) )
results = ( r[ 'results' ] for r in results )
results = ( [ r[ 'name' ], [ 'retype' ], r[ 'count' ], r[ 'source' ] ] for r in results )
```

Filters

We can even filter our data in our generator pipeline.

```
results = ( r for r in results if ([2] > 0 ) # remove results with. count of zero
foo(results) # do something else with your results
```

Lesson 11: String Formatting

Updated 9 months ago by [DELETED] in COMP 3321 (U) Lesson 11: String Formatting

UNCLASSIFIED

(U) Intro to String Formatting

(U) String formatting is a very powerful way to display information to your users and yourself. We have used it through many of our examples, such as this:

```
'This is a formatted String {}' . format ( "------>hi I'm a formatted String argument<-----" )
```

(U) This is probably the easiest example to demonstrate. The empty curly brackets {} take the argument passed into format. (U) Here's a more complicated example:

```
'{2} {1} and {0}' .format ( 'Henry' , 'Bill' , 'Bob' )
```

(U) Arguments can be positional, as illustrated above, or named like the example below. Order

does matter, but names can help.

```
'{who} is really {what}!'.format(who='Tony', what='awesome')
```

(U) You can also format lists:

```
cities = ['Dallas', 'Baltimore', 'DC', 'Austin', 'New York']
'{0[4]} is a really big city.'.format(cities)
```

(U) And dictionaries:

```
lower_to_upper = {'a': 'A', 'b': 'B', 'c': 'C'}
"This is a big letter.0[a]>".format(lower_to_upper)  # notice no quotes around.
"This is a big letter {lookup[a]}".format(lookup=lower_to_upper)  # can be named

for little, big in lower_to_upper.items():
    print('[-->{0:10} -- [1:10}<--]'.format(little, big))
```

(U) If you actually want to include curly brackets in your printed statement, use double brackets

like this: {{ }} .

```
"{{0}} {0}".format('Where do I get printed?')
```

(U) You can also store the format string in a variable ahead of time and use it later:

```
the_way_i_want_it = '{0:>6} = {0:>#16b} = {0:#06x}'

for i in 1, 25, 458, 7890:
    print(the_way_i_want_it.format(i))
```

(U) Format Field Names

(U) Here are some examples of field names you can use in curly brackets within a format string.

{<\field name>)

(U) 1 : the second positional argument

(U) name : keyword argument

(U) 0.var: attribute named var of the first positional argument

(U) 3[0]: element. of the fourth positional argument

(U) me_data[key]: element associated with the specific key string.key' of me_data

(U) Format Specification

(U) When using a format specification, it follows the field name within the curly brackets, and its elements must be in a certain order. This is only for reference; for a full description, see the Python documentation on string formatting.

{<field name>:<format spec>}

1. (U) Padding and Alignment

   : align right - < : align left - = : only for numeric types - ^ : center

2. (U) Sign

   : prefix negative numbers with. minus sign

   : like - but also prefix positive numbers with. +

   : like - but also prefix positive numbers with. space

3. (U) Base Indicator.precede with. hash # like above)

   0b : binary

   0o : octal

   0x : hexadecimal

4. (U) Digit Separator -, : use a comma to separate thousands

5. (U) Field Width

   leading 0 : pad with zeroes at the front

6. (U) Field Type.letter telling which type of value should be formatted)

   s : string.the default)

   b : binary

   d : decimal: base 10

   o : octal

   x : hex uses lower case letters

   x : hex uses upper case

   n : like., use locale settings to determine decimal point and thousands seperator

   no code integer: like d

   e : exponential with small.

   E : exponential with big.

   f : fixed point, nan for not. number and inf for infinity

   F : same as. but uppercase nan and inf

   g : general format

   g : like. but uppercase

   n : locale settings like.

   % : times 100, displays as. with. %

no code decimal: like., precision of twelve and always one spot after decimal point

7. (U) Variable Width

(U) New in Python3.6: f-strings

```
# Add 'f' before the string to create an f-string
# Expression added directLy inside the '{}' brackets rather than after the format statement
x = 34
y = 2
f"34 * 2 = {x*y}"

my_name =  'Bob'
f"My name is{my_name}"
```

(U) Examples

```
'{0:{1}.{2}f}'  .format.9876.5432, 18, 3)
'{0:010.4f}'  .format (-123.456)
'{0:+010.4f}'  .format (-123.456)

for  i  in  range(1, 6):
    print(  '{0:10.{1}f}'  .format(123.456, i))

v = {  'value'  :876.543,  'width'  :15,  'precision'  :5}

"{0[value]:{0[width]}.{0[precision]}}"  .format(v)

data = [(  'Steve'  , 59, 202), (  'Samantha'  , 49, 156), (  'Dave'  , 61, 135)]

for  name, age, weight  in  data:
    print(  '{0:<12s} {1:4d} {2:4d}'  .format(name, age, weight))

# same as above but with f-strings

data = [(  'Steve'  , 59, 202), (  'Samantha'  , 49, 156), (  'Dave'  , 61, 135)]

for  name, age, weight  in  data:
    print(f  '{name:<12s> {age:4d} {weight:4d}'  )
```

UNCLASSIFIED

COMP3321 Day01 Homework - GroceryList

Updated almost 3 years ago by [DELETED] in COMP 3321 (U) Homework for DayOl of COMP3321.

Task is to sort items into bins.

##GroceryList

```
myGroceryList = [ "apples" , "bananas" , "milk" , "eggs" , "bread" ,
                  "hamburgers" , "hotdogs" , "ketchup" , "grapes" ,
                  "tilapia" , "sweet potatoes" , "cereal" ,
                  "paper plates" , "napkins" , "cookies" ,
                  "ice cream" , "cherries" , "shampoo" ]

## Items by category
vegetables = [ "sweet potatoes" , "carrots" , "broccoli" , "spinach" ,
"onions" , "mushrooms" , "peppers" ]
fruit = [ "bananas" , "apples" , "grapes" , "plumbs" , "cherries" , "pineapple" ]
cold_items = [ "eggs" , "milk" , "orange juice" , "cheese" , "ice cream" ]
proteins = [ "turkey" , "tilapia" , "hamburgers" , "hotdogs" , "pork chops" , "ham" , "meatballs" ]
boxed_items = [ "pasta" , "cereal" , "oatmeal" , "cookies" , "ketchup" , "bread" ]
paper_products = [ "toilet paper" , "paper plates" , "napkins" , "paper towels" ]
toiletry_items = [ "toothbrush" , "toothpaste" , "deodorant" , "shampoo" , "soap" ]

## My items by category

my_vegetables = []
my_fruit = []
my_cold_items = []
my_proteins = []
my_boxed_items = []
my_paper_products = []
my_toiletry_items = []
```

(U) Fill in your code below. Sort the items in myGroceryList by type into appropriate my_category

lists using looping and decision making

```
print( "My vegetable list: " , my_vegetables)
print( "My fruit list: " , my_fruit)
print( "My cold item list: " , my_cold_items)
print( "My protein list: " , my_proteins)
print( "My boxed item list: " , my_boxed_items)
print( "My paper product list: " , my_paper_products)
print( "My toiletry item list: " , my_toiletry_items)
```

Dictionary and File Exercises

Updated over 2 years ago by [DELETED] in COMP 3321 (U) Dictionary and file exercises for

COMP3321.

##Lists and Dictionary Exercises ##Exercise 1 (Euler's multiples of 3 and 5 problem) If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9 The sum of these multiples is 23.

Find the sum of all the multiples of. or. below 1000.

```
multiples_3 = [i for i in range(3,1000,3)]
multiples_5 = [i for i in range(5,1000,5)]
multiples = set((multiples_3 + multiples_5)) # set will remove dupLicate numbers
sum(multiples) # add all the numbers together

# you can also do this in one line:
sum([i for . in range(3,1000) if ( % 3 == 0 or .% 5 == 0])
```

Exercise 2

Write a function that takes a list as a parameter and returns a second list composed of any objects that appear more than once in the original list

duplicates([1,2,3,6,7,3,4,5,6]) should return.3,6]

what should duplicates(['cow','pig','goat','horse','pig']) return?

```
# you can use a dictionary to keep track of the number of times seen
def duplicates (x):
    dup={}
    for i in x:
        dup[i] = dup.get(i,0)+1
    result = []
    for i in dup.keys():
        if dup[i] > 1:
            result.append(i)
    return result
    x = [1,2,3,6,7,3,4,5,6]
    duplicates(x)

#you can also just use Lists…
def duplicates2 (x) :
    dup = []
    for . in .:
        if (x.count(i) > 1 and x not in dup:
            dup.append(i)
    return dup

y = [ 'cow' , 'pig' , 'goat' , 'horse' , 'pig' ]
duplicates2(y)

z = [ '2016' , '2015' , '2014' ]
```

```
duplicates(z)
```

Exercise 3

Write a function that takes a portion mark as input and returns the full classification

convert_classification('U//FOUO') should return.UNCLASSIFIED //FOR OFFICIAL USE ONLY'

convert_classification('S//REL TO USA, FVEY') should return.SECRET//REL TO USA, FVEY'

```
# just create a "Lookup tabLe" for potenial portion marks
full_classifications = {  'U//FOUO'  :  'UNCLASSIFIED//~~FOR OFFICIAL USE ONLY~~'  ,
                          'C//REL TO USA, FVEY'  :  'CONFIDENTIAL//REL TO USA, FVEY'  ,
                          'S//REL TO USA, FVEY'  :  'SECRET//REL TO USA, FVEY'  ,
                          'S//SI//REL TO USA, FVEY'  :  'SECRET//SI//REL TO USA, FVEY'  ,
                          'TS//REL TO USA, FVEY'  :  'TOP SECRET//REL TO USA, FVEY'  ,
                          'TS//SI//REL TO USA, FVEY'  :  'TOP SECRET//SI//REL TO USA, FVEY'  }
def  convert_classification  (x) :
    return  full_classifications.get(x,  'UNKNOWN'  )  # Look up the value for the portion mark
convert_classification(  'U//FOUO'  )
convert_classification(  'S//REL TO USA, FVEY'  )
convert_classification(  'C//SI'  )
```

File Input/Output Exercises

These exercises build on concepts in Lesson 3 (Flow Control, e.g., for loops) and Lesson 4 (Container Data Type, e.g, dictionaries). You will use all these concepts together with reading and writing from files

First, Get the Data

Copy the sonnet from https://urn.nsa.ic.gov/t/tx6qm . and paste it into. new text file named sonnet.txt.

Exercise 1

Write a function called file_capitalize() that takes an input file name and an output file name, then writes each word from the input file with only the first letter capitalized to the output file. Remove all punctuation except apostrophe.

```
file_capitalize(  'sonnet.txt'  ,  'sonnet_caps.txt'  )  # => capitalized words written to sonnet_caps.txt
# use heip('') to see what each of these string methods are doing
def  capitalize  ( sentence):
```

```
    words = sentence.split ()                    # use spLit to split the string by spaces.i-e.j words)
    new_words = [ word.strip().capitalize( )                    for    word    in    words ]        # captialize each word
    return    ' '    . join(new_words)                # create and return one string by combing words with ' '

def    remove_punct        (sentence):
    # since replace() method returns. new stringj you can chain calls to the replace()
    # method in order to remove all punctuation in one line of code
    return    sentence.replace(        '.'    ,    ''    ).replace(        ','    ,    ''    ).replace(        ':'    ,  ''    ).replace(        ';'    ,  ''    )

def    file_capitalize            (infile_name, outfile_name):
    infile = open(infile__name.                    'r'    )    # open the input file
    outfile = open(outfile_name,                    'w'    )    # open the outpu file
    for    line    in    infile:        # loop through each line of input
        outfile.write(capitalize(remove_punct(line)) +                    '\n'    )    # write the capitalized version to the output file
    infile.close()            # finallyj close the files
    outfile.close()
file_capitalize(            'sonnet.txt'        ,    'sonnet_caps.txt'            )
```

Exercise 2

Make a function called file_word_count() that takes a file name and returns a dictionary containing

the counts for each word. Remove all punctuation except apostrophe. Lowercase all words.

```
file_word_count(            'sonnet.txt'        )    # => { 'it': 4, 'me': 1, … }
def    file_word__count            (infile_name):
    word_counts = {}
    with    open(infile_name,            'r'    )    as    infile:        # using 'with' so we don't have to cLose the file
    for    line    in    infile:        # Loop over each Line in the fiLe
        words * remove_punct(line)                # we can use the remove_punct from exercise above
        words = words.split()                # split the Line into words
    for    word    in    words:        # Loop over each word
        word = word.strip().lower()
        # add one to the current count for the word.start at. if not there)
        word_counts[word] = word_counts.get(word, 0) + 1

    return    word_counts        # return the whoLe dictionary of word counts
counts = file_word_count(            'sonnet.txt'        )
counts
```

Extra Credit

Write the counts dictionary to a file, one key:value per line.

```
def    write_counts        (outfile_name, counts):
    with    open(outfile_name,            'w'    , encoding=        'utf-8'        )    as    outfile:
    # to Loop over a dictionary, use the items() method
```

```
    # items() will return. 2-element tupLe containing a key and a value
    # below we pull out the values from the tuple into their own variables, word and count
    for word, count in counts.items():
        outfile.write(word + + str(count) + '\n') # write out in key:value format

write_counts( 'sonnet_counts.txt' , counts) # use the counts dictionary from Exercise 2 above
```

Structured Data and Dates Exercise

Updated over 3 years ago by [DELETED] in COMP3321 (U) COMP3321 exercise for working with structured data and dates.

Structured Data and Dates Exercise

Save the Apple stock data from https://urn.nsa.ic.gov/t/Ogrli to aapl.csv.

Use DictReader to read the records. Take the daily stock data and compute the average adjusted close ("Adj Close") per week. Hint: Use .isocalendar() for your datetime object to get the week number.

For each week, print the year, month, and average adjusted close to two decimal places.

```
# Year 2015, Week 23, Average Close 107.40
# Year 2015, Week 22, Average Close 105.10
from csv import DictReader
from datetime import datetime

def average (numbers):
    if len(numbers) == 0:
        return 0.0
    return sum(numbers) / float(len(numbers))

def get_year_week (record):
    dt = datetime.strptime(record[ 'Date' ], '%Y-%m-%d' )
    return (dt.year, dt.isocalendar()[1])

def get_averages (data) :
    avgs = {}
    for year_week, closes in data.items():
        avgs[year_week] = average(closes)
    return avgs

def weekly_summary (reader) :
    weekly_data = {}

    for record in reader:
```

```
            year_week = get_year_week(record)
            if year_week not in weekly_data:
                weekly_data[year_week] = []
            weekly_data[year_week].append(float(record['Adj Close']))
        return get_averages(weekly_data)

def file_weekly_summary(infile_name):
    with open(infile_name, 'r') as infile:
        return weekly_summary(DictReader(infile))

def print_weekly_summary(weekly_data):
    for year_week in reversed(sorted(weekly_data.keys())):
        year = year_week[0]
        week = year_week[1]
        avg = weekly_data[year_week]
        print('Year {year}, Week {week}, Average Close{avg:.2f}'.format(year=year, week=week, avg=avg))

data = file_weekly_summary('aapl.csv')
print_weekly_summary(data)
```

Extra

Use csv.DictWriter to write this weekly data out to a new CSV file.

```
from csv import DictWriter

def write_weekly_summary(weekly_data, outfile_name):
    headers = ['Year', 'Week', 'Avg']

    with open(outfile_name, 'w', newline='') as outfile:
        writer = DictWriter(outfile, headers)
        writer.writeheader()

        for year_week in reversed(sorted(weekly_data.keys())):
            rec = {'Year':year_week[0], 'Week':year_week[1], 'Avg':weekly_data[year_week]}
            writer.writerow(rec)

data = file_weekly_summary('aapl.csv')
write_weekly_summary(data, 'aapl_summary.csv')
```

Extra Extra

Use json.dumps() to write a JSON entry for each week on a new line,

```
import json
```

```
def write_json_weekly_summary (weekly_data, outfile_name):
    with open(outfile_name, 'w') as outfile:
        for yearweek in reversed(sorted(weekly_data . keys())):
            rec = { 'year' :year_week[0], 'week' :year_week[l], 'avg' :weekly_data[year_week]}
            outfile.write(json.dumps(rec) + '\n' )

data = file_weekly_summary( 'aapl.csv' )
write_json_weekly_summary(data, 'aapl.json' )
```

Datetime Exercise Solutions

Created almost 3 years ago by [DELETED] in COMP 3321 (U) Solutions for the Datetime exercises

(U) Datetime Exercises

(U) How long before Christmas?

```
import datetime, time
print(datetime.date(2017, 12, 25) - datetime.date.today())
```

(U) Or, if you're counting the microseconds:

```
print(datetime.datetime(2017, 12, 25) - datetime.datetime.today())
```

(U) How many seconds since you were born?

```
birthdate = datetime.datetime(1985, 1, 31)
time_since_birth = datetime.datetime.today() - birthdate
print(format(time_since_birth.total_seconds()))
```

(U) What is the average number of days between Easter and Christmas for the years 2000 - 2999?

```
from dateutil.easter import easter
total = 0
span = range(2000, 3000)

for year in span:
    total += (datetime.date(year, 12, 25) - easter(year)).days

average = total / len(span)
print( '{:6.4f}' .format(average))
```

(U) What day of the week does Christmas fall on this year?

```
datetime.date(2015, 12, 25).strftime( '%A' )
```

(U) You get a intercepted email with a POSIX timestamp of 1435074325. The email is from the leader of a Zendian extremist group and says that there will be an attack on the Zendian capitol in 14 hours. In Zendian local time, when will the attack occur? (Assume Zendia is in the same time zone as Kabul)

```
import pytz
utc_tz = pytz.timezone( 'Etc/UTC' )
email_time_utc = datetime.datetime.fromtimestamp(1435074325, tz=utc_tz)
attack_time_utc = email_time_utc + datetime.timedelta(hours=14)
zendia_tz = pytz.timezone( 'Asia/Kabul' )
attack_time_zendia = attack_time_utc.astimezone(zendia_tz)

print(email_time_utc)
print(attack_time_utc)
print(attack_time_zendia)
```

Object Oriented Programming and Exercise

Created over 3 years ago by [DELETED] in COMP 3321 (U) COMP3321 exercise for object oriented programming and exceptions.

Object Oriented Programming and Exceptions Exercise

Make a class called Symbol that holds data for a stock symbol, with the following properties:

```
self.name
self.daily_data
```

It should also have the following functions:

```
def __init__(self, name, input_file)
def data_for_date(self, date_str)
```

init(self, name, input_file) should open the input file and read it with DictReader, putting each entry in self.daily_data, using the date strings as the keys. Make sure to open the daily data file within a try/except block in case the file does not exist. If the file does not exist, set self.daily_data to an empty dictionary.

data_for_date(self, date_str) should take. date string and return the dictionary containing that days' data. If there is no entry for that date, return an empty dictionary.

Tests

Make sure the following execute as specified in each comment. You can get the aapl.csv file from https://urn.nsa.ic.gov/t/Ogrli . The apple.csv file should not exist.

```
sl = Symbol ( 'AAPL' , 'aapl.csv' )
print(si.data_for_date( '2015-08-10' )) # should return. dictionary for that date
print(sl.data_for_date( '2015-08-09' )) # should return an empty dictionary

s2 = Symbol ( 'AAPL' , 'apple.csv' ) # should not raise an exception!
print(s2.data_for_date( '2015-08-10' )) # should return an empty dictionary
print(s2.data_for_date( '2015-08-09' )) # should return an empty dictionary
```

Module: Collections and Itertools

Updated almost 2 years ago by [DELETED] in COMP 3321 (U) Module: Collections and Itertools

(U) Any programming language has to strike a balance between the number of basic elements it exposes, like control structures, data types, and so forth, and the utility of each one. For example, Python could do without tuples entirely, and could replace the dict with a list of lists or even a single list where even-numbered indices contain keys and odd-numbered indices contain values. Often, there are situations that happen so commonly that they warrant inclusion, but inclusion in the builtin library is not quite justified. Such is the case with the collections and itertools modules. Many programs could be simplified with a defaultdict, and having one available with a single from collection import defaultdict is much better than reinventing the wheel every time it's needed.

(U) value Added Containers with collections

(U) Suppose we want to build an index for a poem, so that we can look up the lines where each word occurs. To do this, we plan to construct a dictionary with the words as keys, and a list of line numbers is the value. Using a regular dict, we'd probably do something like this:

```
poem = """mary had a little lamb
it's fleece was white as snow
and everywhere that mary went
the lamb was sure to go"""

index = {}

for linenum, line in enumerate(poem.split( '\n' )):
```

```
    for word in line.split():
        if word in index:
            index[word].append(linenum)
        else :
            index[word] = [linenum]
```

(U) This code would be simpler without the inner if … else … clause. That's exactly what a defaultdict is for; it takes a function (often a type, which is called as a constructor without arguments) as its first argument, and calls that function to create a default value whenever the program tries to access a key that isn't currently in the dictionary. (It does this by overriding the missing. method of dict .) In action, it looks like this:

```
from collections import defaultdict

index = defaultdict(list)

for linenum, line in enumerate(poem. split( '\n' )):
    for word in line.split():
        index[word].append(linenum)
```

(U) Although a defaultdict is almost exactly like a dictionary, there are some possible complications because it is possible to add keys to the dictionary unintentionally, such as when testing for membership. These complications can be mitigated with the get method and the in operator.

```
'sheep' in index # False
1 in index.get( 'sheep' ) # Error
'sheep' in index # still False
2 in index[ 'sheep' ] # still False, but …
'sheep' in index # previous statement accidentally added ' sheep'
```

(U) You can do crazy things like change the default_factory.it's just an attribute of the defaultdict object), but it's not commonly used:

```
import itertools

def constant_factory (value) :
    return itertools.repeat(value).__next__

d = defaultdict(constant_factory( '<missing>' ))

d.update(name= 'John' , action= 'ran' )

'{0[name]} {0[action]} to {0[object]}' .format(d)

d # "object" added to d
```

(U) A Counter is like a defaultdict(int) with additional features. If given a list or other iterable when constructed, it will create counts of all the unique elements it sees. It can also be constructed from a dictionary with numeric values. It has a custom implementation of update and some specialized methods, like most_common and subtract.

```
from collections import Counter
word_counts = Counter(poem.split())
word_counts . most_common( 3 )
word_counts. update( 'lamb lamb lamb stew' .splitQ)
word_counts.most_common(3)
c = Counter(a=3, b=l)
d = Counter(a=l, b=2)
c + d
c - d # Did you get the output you expected?
(c - d) + d
c & d
c | d
```

(U) An OrderedDict is a dictionary that remembers the order in which keys were originally inserted, which determines the order for its iteration. Aside from that, it has a popitem method that can pop from either the beginning or end of the ordering. (U) namedtuple is used to create lightweight objects that are somewhat like tuples, in that they are immutable and attributes can be accessed with [] notation. As the name indicates, attributes are named, and can also be accessed with the . notation. It is most often used as an optimization, when speed or memory requirements dictate that a dict or custom object isn't good enough. Construction of a namedtuple is somewhat indirect, as namedtuple takes field specifications as strings and returns a type, which is then used to create the named tuples, named tuples can also enhance code readability.

```
from collections import namedtuple
Person = namedtuple( 'Person' , 'name age gender' )
bob = Person(name= 'Bob' , age=30, gender= 'male' )
print( '%s is a %d year-old %s' % bob ) # 2.x style string formatting
```

```
print(      '{} is a {} year-old {}'                    .format (*bob) )
print(      '%s is a %d year-old %s'                      % (bob.name, bob.age, bob.gender) )
print(      '{} is a {} year-old {}'                    .format(bob.name, bob.age, bob.gender) )
bob[0]
bob[    'name'      ]    ## TypeError
bob.name
print(      '%(name)s is a %(age)d year-old %(gender)s'                                  % bob )       # Doesn't work
print(      '{name} is a {age} year-old.gender}'                                .format(*bob) )       # Doesn't work
print(      '{0.name} is a {0.age} year-old {0.gender}'                                  .format(bob) )       # Marks!
```

(U) Finally, deque provides queue operations.

```
from      collections              import       deque
d = deque(          'ghi'       )          # make a new deque with three items
d.append(          'j'     )              # add a new entry to the right side
d.appendleft(                'f'       )    # add a new entry to the left side
d.popleft()                               # return and remove the leftmost item
d.rotate(1)                               # right rotation
d.extendleft(                'abc'       )  # extendLeft() reverses the input order
```

(U) The collections module also provides Abstract Base classes for common Python interfaces. Their purpose and use is currently beyond the scope of this course, but the documentation is reasonably good.

(U) Slicing and Dicing with itertools

Given one or more lists, iterators, or other iterable objects, there are many ways to slice and dice the constituent elements. The itertools module tries to expose building block methods to make this easy, but also tries to make sure that its methods are useful in a variety of situations, so the documentation contains a cookbook of common use cases. We only have time to cover a small subset of the itertools functionality. Methods from itertools usually return an iterator, which is great for use in loops and list comprehensions, but not so good for inspection; in the code blocks that follow, we often call list on these things to unwrap them. (U)The chain method combines iterables into one super-iterable. The groupby method separates one iterator into groups of

adjacent objects, possibly as determined by an optional argument-this can be tricky, especially because there's no look back to see if a new key has been encountered previously.

```
import itertools
list(itertools.chain(range(5),[5,6])) == [0,1,2,3,4,5,6]
size_groups = itertools.groupby([1,1,2,2,2,'p','p',3,4,3,3,2])
[(key, list(vals)) for key, vals in size_groups]
```

(U) A deeply nested for loop or list comprehension might be better served by some of the combinatoric generators like product, permutations, or combinations.

```
iter_product = itertools.product([1,2,3],['a','b','c'])
list(iter_product)
iter_combi = itertools.combinations("abcd",3)
list_combi = list(iter_combi)
list_combi
iter_permutations = itertools.permutations("abcd",3)
list.iter_pe mutations)
```

(U) itertools can also be used to create generators:

```
counter = itertools.count(0, 5)
next(counter)
print(list(next(counter) for c in range(6)))
```

(U) Be careful… What's going on here?!?

```
counter = itertools.count(0.2,0.1)

for c in counter:
    print(c)
    if ( > 1.5:
        break
        cycle = itertools.cycle('ABCDE')

for c in range(10):
    print(next(cycle))
    repeat = itertools.repeat('again!')

for i in range(5):
    print(next(repeat))
    repeat = itertools.repeat('again!', 3)

for i in range(5):
    print(next.repeat))
```

```
nums = range(10,0,-1)
my_zip = zip(nums, itertools.repeat(                    'p'    ))
for    thing    in    my_zip:
        print(thing)
```

Functional Programming

Created over 3 years ago by [DELETED] in COMP 3321 (U //FOUO) A short adaptation of [DELETED] supplement "A practical introduction to functional programming" in Python to COMP 3321 materials. Also discusses lambdas.

UNCLASSIFIED

(U) Introduction

(U) At a basic level, there are two fundamental programming styles or paradigms:

- imperative or procedural programming and
- declarative or functional programming.

(U) Imperative programming focuses on telling a computer how to change a program's state--its stored information--step by step. Most programmers start out learning and using this style. It's a natural outgrowth of the way the computer actually works. These instructions can be organized into functions/procedures (procedural programming) and objects (object-oriented programming), but those stylistic improvements remain imperative at heart.

(U) Declarative programming, on the other hand, focuses on expressing what the program should do, not necessarily how it should be done. Functional programming is the most common flavor of that. It treats a program as if it is made up of mathematical-style functions: for a given input x, running it through function f will always give you the same output f(x), and x itself will remain unchanged afterwards. (Note that this is not necessarily the same as a procedural-style function, which may have access to global variables or other "inputs" and which may be able to modify those inputs directly.)

(U) TL;DR

(U) The key distinction between procedural and functional programming is this: a procedural function may have side effects--it may change the state of its inputs or something outside itself, giving you a different result when running it a second time. Functional programming avoids side effects, ensuring that functions don't modify anything outside themselves.

(U) Note

(U) The contents of this notebook have been borrowed from the beginning of [DELETED] essay. "A practical introduction to functional programming." A full notebook version of that essay can be found here. (Note that it uses Python 2.)

(U) Functional vs. Not

(U) The best way to understand side effects is with an example. (U) This function is not functional:

```
a = 0
def increment():
    global a
    a += 1
```

(U) This function is functional:

```
def increment(a):
    return a + 1
```

(U) Map-Reduce

(U) Let's jump into functional coding. One common use is map-reduce, which you may have heard of. Let's see if we can make sense of it.

(U) map

(U) Conceptually, map is a function that takes two arguments: another function and a collection of items. It will

1. run the function on each item of the original collection and
2. return a new collection containing the results,
3. leaving the original collection unchanged. (U) In Python 3, the input collection must simply be iterable (e.g. list, tuple, string). Its map function returns an iterator that runs the input function on each item of iterable.

(U) Example 1: Name Lengths

(U) Take a list of names and get a list of the name lengths:

```
name_lengths = map(len, [ "Mary" , "Isla" , "Sam" ])
print(list(name_lengths))
```

(U) Example 2: Squaring

(U) Square every number in a list:

```
squares = map( lambda x: x * x, [0, 1, 2, 3, 4])
print(list(squares))
```

(U) A digression on lambda

(U) So what's going on with that input function? lambda will let you define and use an unnamed function. Arguments fit between the lambda and the colon while the stuff after the colon gets implicitly returned (i.e. without explicitly using. return statement). (U) Lambdas are most useful when:

your function is simple and

you only need to use it once. (U) Consider the usual way of defining. function:

```
def square (x):
    return x * x

square(4)

# we could have done this instead

squares = map(square, [0, 1, 2, 3, 4])
print(list(squares))
```

(U) Now let's define the same function using. lambda:

```
lambda x: x * x
```

(U) Fine, but how do we call that resulting function? Unfortunately, it's too late now; we didn't store the result, so it's lost in the ether. (U) Let's try again:

```
ima_function_variable = lambda x: x * x
type(ima_function_variable)
ima_function_variable(4)
```

```
# be careful!
ima_function_variable =  'something else'

# our Lambda function is gone again
ima_function_variable(4)
```

(U) Example 4: Code Names

(U) OK, back to map. Here's a procedural way to take a list of real names and replace them with randomly assigned code names,

```
import random
names = [ 'Mary' , 'Isla' , 'Sam' ]
code_names = [ 'Mr. Pink' , 'Mr. Orange' , 'Mr. Blonde' ]

for i in range(len(names)):
    names[i] = random.choice(code_names)

print(names)
```

(U) Here's the functional version:

```
names = [ 'Mary' , 'Isla' , 'Sam' ]
covernames = map( lambda x: random.choice([ 'Mr. Pink' , 'Mr. Orange' , 'Mr. Blonde' ]), names)
print(list(covernames) )
```

(U) Exercise: Code Names…Improved?

(U) The procedural code below generates code names using a new method. Rewrite it using map.

```
names = [ 'Mary' , 'Isla' , 'Sam' ]

for i in range(len(names)):
    names[i] = hash(names[i])

print(names)
# your code here
```

reduce

(U) Reduce is the follow-on counterpart to map. Given a function and a collection of items, it uses the function to combine them into a single value and returns that result. (U) The function passed to

reduce has some restrictions, though. It must take two arguments: an accumulator and an update value. The update value is like it was before with map; it will get set to each item in the collection one by one. The accumulator is new. It will receive the output from the previous function call, thus "accumulating" the combined value from item to item through the collection. (U) Note: in Python 2, reduce was a built-in function. Python 3 moved it into the functools package.

(U) Example

(U) Get the sum of all items in a collection.

```
import functools

sum = functools.reduce( lambda a, x: a + x, [0, 1, 2, 3, 4])
print(sum)
```

UNCLASSIFIED

Recursion Examples

Updated over 3 years ago (U) Some simple recursion examples in Python

Recursion

Recursion provides a way to loop without loops. By calling itself on updated data, a recursive function can progress through a problem and traverse the options.

Nth Fibonacci Number

https://wikipedia.nsa.ic.gov/en/Fibonacci_number This returns the nth Fibonacci number in the Fibonacci Sequence using recursion.

```
def nth_fibonacci (n) :
    if n < 1:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 1
    else :
        return nth_fibonacci(n-2) + nth_fibonacci(n-1)

nth_fibonacci(10)
```

Fibonacci Sequence

This returns a list of the first n Fibonacci Numbers using recursion.

```
def fibonacci (n, seq=[]):
    if len(seq) == n:
        return seq
    elif len(seq) == 0:
        return fibonacci(n, [1])
    elif len(seq) == 1:
        return fibonacci(n, [1,1])
    else :
        next_value = seq[-2] + seq[-l]
        return fibonacci(n, seq + [next_value])

fibonacci(5)
```

Simple Game

This simple game just takes in a list of nine elements and tries to modify each slot until all the numbers from 1 to 9 are in the list.

```
import random

def improve (input_list, missing):
    random_index = random.choice(list(range(len(input_list))))
    random_value = random.choice(missing)
    new_list = input_list[:]
    new_list[random_index] = random_value
    return new_list

def find_missing (input_list):
    missing = [ x for x in list(range(1,10)) if x not in input_list ]
```

```
    return missing

def one_to_nine(input_list):
    print(input__list)
    missing = find_missing(input_list)
    if len(missing) == 0:
        return input_list
    else:
        new_list = improve(input_list, missing)
        return one_to_nine(new_list)

one_to_nine( [1,1,1,1,1,1,1,1,!])
```

Simple Game Revised

This revision of the same simple game comes up with a list of possible improvements and tries to pick the best one to pursue. You'll notice that it takes fewer attempts to reach an anser than the original version of this simple game.

```
def improve(input_list, missing):
    random_index = random.choice(list(range(len(input_list))))
    random_value = random.choice(missing)
    new_list = input_list[:]
    new_list[random_index] = random_value
    return new_list

def find_missing(input_list):
    missing = [ x for x in list(range(1, 10) ) if x not in input_list ]
    return missing

def score(input__list):
    missing = find_missing(input_list)
    return (len(missing), input_list)

def best_scoring_list(scored_lists):
    lowest = 100
    best_list = []
    for x in scored_lists:
        score = x[0]
        input_list = x[1]

        if score < lowest:
            lowest = score
            best_list = input_list
    return best_list

def one_to_nine(input_list):
    print(input_list)
    missing = find_missing(input_list)
```

```
    if len(missing) == 0:
        return input_list
    else:
        possible_improvements = [ improve(input_list, missing) for i in range(len(missing))]
        scored__improvements = [ score(i) for i in possible_improvements ]
        best_list = bestscoring_list(scored_improvements)
        return one_to_nine(best_list)

one_to_nine([1,1,1,1,1,1,1,1,1])
```

Simple Game as a Tree

We can think of our strategy as a tree of options that we traverse, following branches that show that they're going to improve our chances of finding a solution. This is an extremely simplified form of what many video games use for their Al. We put our possible_improvements in a generator so they will only be created as needed. If we put them in a list comprehension as before, then all possible improvements would be generated even though many of them will likely go unused. In the end, we return any() with a generator for the branches. Since any() only needs one item to be True, it will return True as soon as a solution is found; when len(missing) == 0. You'll notice that this results in more iterations of one_to_nine() than in the previous revision. However, the previous revision also generated a lot of data that ends up getting discarded. In other words, there's probably more processing and memory consumed by the previous revision behind the scenes.

```
def improve (input_list, missing):
    random_index = random.choice(list(range(len(input_list))))
    random_value = random.choice(missing)
    new_list = input_list[:]
    new_list[random_index] = random_value
    return newlist

def find_missing (input_list):
    missing = [ x for x in list(range(1, 10)) if x not in input_list ]
    return missing

def one_to_nine (input_list, prev_missing=None):
    print(input_list)
    missing = find_missing(input_list)
    if prev_missing is None:
        return one_to_nine(input_list, missing)
    elif len(missing) == 0:
        return True
    elif len(missing) > len(prev_missing):
        return False
    else:
        possible_improvements = ( improve(input_list, missing) for i in range(len(missing)) )
    return any(( one_to_nine(p, missing) for p in possible_improvements ))
```

```
one_to_nine([1,1,1,1,1,1,1,1,1])
```

Module: Command Line Arguments

Updated almost 2 years ago by [DELETED] in COMP 3321 (U) Module: Command Line Arguments

UNCLASSIFIED

(U) Most command line programs accept options and arguments, and many even provide help messages that indicate what options are available, and how they are to be used. For example, the utility program mv takes two arguments, and most often moves the first argument (the source) to the second (the destination). It has other ways of operating, which are enabled by optional flags and arguments; from a command prompt, type mv --help to see more. (U) There are several ways to enable this type of functionality in a Python program, and the best way to do it has been a source of contention. In particular, this lesson will cover the argparse module, which was added to the standard library in Python 2.7, and not the optparse module which was deprecated at that time. (U) Everything passed as arguments to a Python program is available in the interpreter as the list of strings in sys.argv. In an interactive session, sys.argv always starts out as [' ']. When running a script, sys.argv[0] is the name of the script. We start by examining what sys.argv looks like. Put the following commands in a file called argtest.py or similar:

```
import sys
print(sys.argv)

# …or make python do it!
contents = ''' import sys
print(sys.argv)
'''
with open ( 'argtest.py' , 'w' ) as f:
    f.write(contents)
```

(U) Close the file and execute it from the command line with some arguments:

```
# the '!' at the beginning tells jupyter to send what follows to the command Line
!python3 argtest.py -xzf --v foo --othervar=bar filel file2
# => ['argtest.py', '-xzf', '--v', 'foo', '--othervar=bar', 'filel', 'file2']
```

(U) In all of the argument parsing that follows, sys.argv will be involved, although that may happen either implicitly or explicitly. Although it is often unwise to do so within a script, sys.argv can be modified, for instance during testing within an interactive session. (U) Note that in Jupyter you still have argv, but it may not be what you expect. If you look at it, you'll see how this Python 3 kernel

is being called:

```
import sys
print(sys.argv)
```

(U) The Hard Way: getopt

(U) For programs with only simple arguments, the getopt module provides functionality similar to the getopt function in C. The main method in the module is getopt, which takes a list of strings, usually sys.argv[1:] and parses it according to a string of options, with optional long options, which are allowed to have more than one letter; explanations are best left to examples. This method returns a pair of lists, one containing (option, value) tuples, the other containing additional positional arguments. These values must then be further processed within the program; it might be useful, for instance, to put the (option, value) tuples into a dict. If getopt receives an unexpected option, it throws an error. If it does not receive all the arguments it requests, no error is thrown, and the missing arguments are not present in the returned value.

```
import getopt

getopt.getopt( '-a arg' .split(), 'a:' ) # a expects an argument
getopt.getopt( '-a arg' .split(), 'a:b' ) # no b, no problem
getopt.getopt( '-b arg -a my-file.txt' .split(), 'ab:' ) # my-fiLe.txt is argument, not option
getopt.getopt( '-a arg --output=other-file.txt my-file.txt' .split(), 'a:b' ,[ 'output=' ]) # Long options
```

(U) For programs that use getopt, usage help must be provided manually.

```
def usage ():
    print( """usage: my_program.py -[abh] filel, file2, ...""" )
# this won't actually find anything in Jupyter, since ipython3 probably doesn't have these options
opts, args = getopt.getopt(sys.argv[1:], 'abh' )
opt_dict = dict(opts)
if '-h' in opt_dict:
    usage()
```

(U) The argparse Module

(U) Integrated help is one of the benefits of argparse, along with the ability to specify both short and long options versions of arguments. There is some additional complication in setting up argparse, but it is the right thing to do for all but the most simple programs.

(U) Basic Usage

(U) The main class in argparse is ArgumentParser. After an ArgumentParser is instantiated, arguments are added to it with the add_argument method. After all the arguments are added, the parse_args method is called. By default, it reads from sys.argv[1:], but can also be passed a list of strings, primarily for testing. Both positional arguments (required) and optional arguments indicated by flags are supported. An example will illustrate the operation.

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('n')
parser.add_argument('-f')
parser.add_argument('-i', '--input')
parser.print_help()
parser.parse_args('abc -f xyz'.split())
parser.parse_args('-f xyz abc --input=myfile.txt'.split())
parser.parse_known_args('-f xyz abc --input=myfile.txt.o otherfile.txt'.split())
args = parser.parse_args('-f xyz abc --input=myfile.txt'.split())
args.f
```

(U) As seen in the final two lines, positional arguments and optioned arguments can come in any order, which is not the case with getopt. If multiple positional arguments are specified, they are parsed in the order in which they were added to the ArgumentParser. The object returned by parse_args is called a Namespace, but it is just an object which contains all the parsed data. Unless otherwise specified, the attribute names are derived from the option names. Positional arguments are used directly, while short and long flags have leading hypens stripped and internal hyphens converted to underscores. If more than one flag is specified for an argument, the first long flag is used if present; otherwise, the first short flag is used.

(U) Here is how argparse could look in your code.

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('n')
parser.add_argument('-f')
parser.add_argument('-i', '--input')
args = parser.parse_args()

print(args)
```

```
contents = '''import argparse
parser = argparse.ArgumentParser()
parser.add_argument('n')
parser.add_argument('-f')
parser.add_argument('-i','--input')
args = parser.parse_args()
print(args)
'''

with open('argparsetest.py', 'w') as f:
    f.write(contents)
```

(U) Now we can simulate running it:

```
!python3 argparsetest.py -f xyz abc --input=myfile.txt
!python3 argparsetest.py -h
```

(U) Advanced Options

(U) The add_argument method supports a large number of keyword arguments that configure behavior more finely than the defaults. For instance, the type argument will make the parser attempt to convert arguments to the specified type, such as int, float, or file. In fact, you could use any class for the type, as long as it can be constructed from. single string argument.

```
parser = argparse.ArgumentParser()
parser.add_argument('n', type=int)
parser.parse_args('5'.split())
```

(U) The nargs keyword lets an argument specify a fixed or variable number of arguments to consume, which are then stored into a list. This applies to both positional and optional arugments. Giving nargs the value '?' makes positional arguments optional, in which case the default keyword is required.

```
parser = argparse.ArgumentParser()
parser.add_argument('n', nargs=2)
parser.add_argument('-m ', nargs='*') # arbitrary arguments
parser.parse__args('n1 n2 -m a b c'.split())
parser.add_argument('o', nargs='?', default='OoO')
parser.parse_args('n1 n2 oOo -m a b c'.split())
```

(U) The default keyword can also be used with optional arguments. When an optional argument is always used as as aflag without parameters, it is also possible to use the action='store_const ' and

const keywords. In this case, when the option is detected, the Namespace is given an appropriately-named attribute with const as its value. If the option is not present in the parsed args, the attribute is created with the value given in default, or None if default isn't set.

```
parser = argparse.ArgumentParser()
parser.add_argument('-n', action='store_const', const=7)
parser.add_argument('-b', action='store_true')
parser.add_argument('-c', action='store_const', const=5, default=3)
parser.parse_args([])
parser.parse_args('-n -b'.split())
parser.parse_args('-cbn'.split())
```

(U)The action keyword can take other arguments; for instance, action='store_true' and action='store_false' can be used instead of setting const to a boolean value. (U) Once again, we have only scratched the surface of a module, argparse in this case. Check out the documentation for more details (e.g. changing attribute names with the dest keyword, writing custom action functions, providing different parsers for subprograms).

UNCLASSIFIED

Module: Dates and Times

Updated 10 months ago by [DELETED] in COMP 3321 (U) How to manipulate dates and times in Python.

UNCLASSIFIED

(U) Introduction

(U) There are many great built-in tools for date and time manipulation in Python. They are spread over a few different modules, which is a little annoying. (U) That being said, the datetime module is very complete and the most useful, so we will concentrate on that one that most. The other one that has some nice methods is the time module.

(U) time Module

(U) The time module is handy for its time accessor functions and sleep command. It is most useful when you want quick access to the time but don't need to manipulate it.

```
import time
time.time()    # "epoch" time (seconds since Jan. 1st, 1970)
time.gmtime()
time.localtime()
```

```
time.gmtime() == time.localtime()
time.asctime()                  # will take an optionaL timestamp
time.strftime(          '%c'    )   # many formatting options here
time.strptime(          'Tue Nov 19 07:04:38 2013'                  )
```

(U) The last method you might use from the time module is sleep. (Doesn't this seem out of place?)

```
time.sleep(10)              # argument is a number of seconds
print(      "I'm awake!"            )
```

(U) datetime Module

(U) The datetime module is a more robust module for dates and times that is object-oriented and has a notion of datetime arithmetic. (U) There are 5 basic types that comes with the datetime module. These are:

1. date : a type to store the date (year, month, day) using the current Gregorian calendar,
2. time : a type to store the time (hour, minute, second, microsecond, tzinfo-all idealized, with no notion of leap seconds),
3. datetime : a type to store both date and time together,
4. timedelta : a type to store the duration or difference between two date, time, or datetime instances, and
5. tzinfo : a base class for storing and using time zone information.we will not look at this).

(U) date type

```
import      datetime
datetime.date(2013, 11, 19)
datetime.date.today()
datetime.date.fromtimestamp(time.time())
today = datetime.date.today()
today.day
today.month
today.timetuple()
today.weekday()             # 0 ordered starting from Monday
today.isoweekday()          # 1 ordered starting from Monday
print(today)
today.strftime(         '%d %m %Y'      )
```

(U) time type

```
t = datetime.time(8, 30, 50, 0)
t.hour = 9
t.hour
t.minute
t.second
print(t)
print(t. replace(hour=12))
t.hour = 8
print(t)
```

(U) datetime type

```
dt = datetime.datetime(2013, 11, 19, 8, 30, 50, 0)
print(dt)
datetime.datetime.fromtimestamp(time.time())
now = datetime.datetime.now()
```

(U) We can break apart the datetime object into date and time objects. We can also combine date and time objects into one datetime object.

```
now.date()
print(now.date())
print(now.time())
day = datetime.date(2011, 12, 30)
t = datetime.time(2, 30, 38)
day
t
dt = datetime.datetime.combine(day,t)
print(dt)
```

(U) timedelta type

(U) The best part of the datetime module is date arithmetic. What do you get when you subtract two dates?

```
day1 = datetime.datetime(2013, 10, 30)
day2 = datetime.datetime(2013, 9, 20)
day1 - day2
print(day1 - day2)
print(day2 - day1)
print(day1 + day2)                    # Of course not… that doesn't make sense :)
```

(U) The timedelta type is a measure of duration between two time events. So, if we subtract two

datetime objects (or date or time) as we did above, we get a timedelta object. The properties of a timedelta object are (days, seconds, microseconds, milliseconds, minutes, hours, weeks). They are all optional and set to 0 by default. A timedelta object can take these values as arguments but converts and normalizes the data into days, seconds, and microseconds.

```
from datetime import timedelta
day = timedelta(days=1)
day
now = datetime.datetime.now()
now + day
now - day
now + 300*day
now - 175*day
year = timedelta(days=365)
another_year = timedelta(weeks=40, days=84, hours=23, minutes=50, seconds=600)
year == another_year
year.total_seconds()
ten_years = 10*year
ten_years
```

(U) Conversions

(U) It's easy to get confused when attempting to convert back and forth between strings, numbers, times, and datetimes. When you need to do it, the best course of action is probably to open up an interactive session, fiddle around until you have what you need, then capture that in. well-named function. Still, some pointers may be helpful. (U) Objects of types time and datetime provide strptime and strftime methods for converting times and dates from strings (a.k.a. parsing) and converting to strings (a.k.a. formatting), respectively. These methods employ a custom syntax that is shared across many programming languages.

```
print(ga_dt)
```

(U) Localize the time using the Georgia timezone, then convert to Kabul timezone

```
kabul_tz = pytz.timezone( 'Asia/Kabul' )
kabul_dt = ga_tz.localize(ga_dt).astimezone(kabul_tz)
print(kabul_tz.normalize(kabul_dt))
```

(U) To get a list of all timezones:

```
pytz.all_timezones
```

(U) The arrow package

(U) The arrow package is a third party package useful for manipulating dates and times in a non-naive way (in this case, meaning that it deals with multiple timezones very seamlessly). Examples below are based on examples from "20 Python Libraries You Aren't Using (But Should)" on Safari.

```
import ipydeps
ipydeps.pip( 'arrow' )

import arrow

t0 = arrow.now()
print(t0)

t1 = arrow.utcnow()
print(t1)

difference = (t0 - tl).total_seconds()

print( 'Total difference: %.2f seconds' % difference)

t0 = arrow.now()
t0

t0.date()

t0.time()

t0.timestamp

t0.year

t0.month

t0.day

t0.datetime

tl.datetime

t0 = arrow.now()
t0.humanize()

t0.humanize()

t0 = t0.replace(hours=-3,minutes=10)
t0.humanize()
```

(U) The parsedate module

(U) The parsedate package is a third party package that does a very good job of parsing dates and times from "messy" input formats. It also does good job of calculating relative times from human-sytle input. Examples below are from "20 Python Libraries You Aren't Using (But Should)" on Safari.

```
ipydeps.pip( 'parsedatetime' )
import parsedatetime as pdt
cal = pdt.Calendar()
examples = [
"2016-07-16" ,
"2016/07/16" ,
"2016-7-16" ,
"2016/7/16" ,
"07-16-2016" ,
"7-16-2016" ,
"7-16-16" ,
"7/16/16" ,
]

# print the header
print( '{:30s}{:>30s}' .format( 'Input' , 'Result' ))
print( '=' * 60)

#Loop through the examples list and show that parseDT successfully parses out the date/time based on the messy

for e in examples:
    dt, result = cal.parseDT(e)
    print( '{:<30s}{:>30}' .format( "'" + e + "'" , dt.ctime()))

examples = [
"19 November 1975" ,
"19 November 75" ,
"19 Nov 75" ,
"tomorrow" ,
"yesterday" ,
"10 minutes from now" ,
"the first of January, 2001" ,
"3 days ago" ,
"in four days' time" ,
"two weeks from now" ,
"three months ago" ,
"2 weeks and 3 days in the future" ,
]

#print the time right now for reference
print( 'Now: {}' .format(datetime.datetime.now(). ctime()), end= '\n\n' )

#print the header
print( '{:40s}{:>30s}' . format ( 'Input' , 'Result' ))
```

```
print(          '='         * 70)

#Loop through the exampies List to show how parseDT can successfully determine the date/time based on both mes
# and messy relative time offset inputs
for         e     in       examples:
            dt, result * cal.parseDT(e)
            print(          '{:<40s}{:>30}'                    .format(           ""          + e +          ""        , dt.ctime()))
```

UNCLASSIFIED

COMP3321 Datetime Exercises

Created almost 3 years ago by [DELETED] in COMP 3321 (U) Datetime Exercises for COMP3321

(U) Datetime Exercise

(U) How long before Christmas? (U) How many seconds since you were born? (U) What is the average number of days between Easter and Christmas for the years 2000 - 2999? (U) What day of the week does Christmas fall on this year? (U) You get a intercepted email with a POSIX timestamp of 1435074325. The email is from the leader of a Zendian extremist group and says that there will be an attack on the Zendian capitol in 14 hours. In Zendian local time, when will the attack occur? (Assume Zendia is in the same time zone as Kabul)

Module: Interactive User Input with ipywidgets

Updated almost 2 years ago by [DELETED] in COMP 3321 (U) Covers the ipywidgets library for getting interactive user input in Jupyter

UNCLASSIFIED//FOR OFFICIAL USE ONLY

(U) ipywidgets

(U) ipywidgets is used for making interactive widgets inside your jupyter notebook (U) The most basic way to get user input is to use the python built in input function. For more complicated types of interaction, you can use ipywidgets

```
#input example (not using ipywidgets)
a=input(          "Give me your input: "                          )
print(         "your input was: "                      +a)

import          ipywidgets
from       ipywidgets              import          *
```

interact is the easiest way to get started with ipywidgets by creating a user interface and automatically calling the specified function

```
def       f (x):
          return          x*2

interact(f,x=10)

def       g (check,y):
          print(         "{} {}"         .format(check,y))

interact(g,check=                    True     ,y=     "Hi there!"               )
```

But, if you need more flexibility, you can start from scratch by picking a widget and then calling the functionality you want. Hint: you get more widget choices this way.

```
IntSlider()
w=IntSlider()
w
```

You can explicitly display using IPython's display module. Note what happens when you display the same widget more than once!

```
from       IPython.display                    import          display
display(w)

w.value
```

Now we have a value from our slider we can use in code. But what other attributes or "keys" does our slider widget have?

```
w.max

new_w=IntSlider(max=200)
display(new_w)
```

You can also close your widget

```
w.close()
new_w.close()
```

Here are all the available widgets:

```
Widget.widget_types
```

Numeric: IntSlider, FloatSlider, IntRangeSlider, FloatRangeSlider, IntProgress, FloatProgress, BoundedlntText, BoundedFloatText, IntText, FloatText Boolean: ToggleButton, Checkbox, Valid Selection: Dropdown, RadioButtons, Select, ToggleButtons, SelectMultiple String Widgets: Text, Textarea Other common: Button, ColorPicker, HTML, Image

```
Dropdown(options=[ "1" , "2" , "3" , "cat" ])
bt = Button(description= "Click me!" )
display(bt)
```

Buttons don't do much on their own, so we have to use some event handling. We can define a function with the desired behavior and call it with the buttons on_click method.

```
def clicker (b) :
    print( "Hello World!!!!" )

bt.on_click(clicker)
def f (change):
    print(change[ 'new' ])

w = IntSlider()
display(w)
w.observe(f,names= 'value' )
```

Wrapping Multiple Widgets in Boxes

When working with multiple input widgets, it's often nice to wrap it all in a nice little box. ipywidgets provides a few options for this--we'll cover HBox (horizontal box) and VBox (vertical box).

HBox

This will display the widgets horizontally

```
fruit_list = Dropdown(
options = [ 'apple' , 'cherry' , 'orange' , 'plum' , 'pear' ]
)
fruit_label = HTML(
value = 'Select a fruit from the list:  '
)
fruit_box = HBox(children=[fruit_label, fruit_list])
fruit_box
```

VBox

This will display the widgets (or boxes) vertically

```
num_label = HTML(value = 'Choose the number of fruits:  ' )
num_options = IntSlider(

min=1,
max=20
)

num_box = HBox(children=(num_label, num_options))

type_label = HTML(
value = 'Select the type of fruit:  '
)

type_options = RadioButtons(
options=( 'Under-ripe' , 'Ripe' , 'Rotten' )
)

type_box = HBox(children=(type_label, type_options))
fruit_vbox = VBox(children=(fruit_box, numbox, type_box))
fruit vbox
```

Specify Layout of the Widgets/Boxes

```
form_item_layout = Layout(
display= 'flex' ,
flex_flow= 'row' ,
justify_content= 'space-between' ,
width= '70%' ,
align_items= 'initial' ,
)

veggie_label = HTML(
value = 'Select a vegetable from the list:  ' ,
layout=Layout(width= '20%' , height= '65px' )
```

```
)

veggie_options = Dropdown(
options=[ 'corn' , 'lettuce' , 'tomato' , 'potato' , 'spinach' ],
layout=Layout(width= '30%' , height= '65px' )
)

veggie_box = HBox(children=(veggie_label, veggie_options),
                  layout=Layout(width= '100%' , border= 'solid 1px'
                  height= '100px' ))
veggie_box
```

Retrieving values from a Box

```
box_values = {}
# the elements in a box can be accessed using the children attribute
for index, box in enumerated(fruit_vbox.children):
    for child in box.children:
        if type( child) != ipywidgets.widgets.widget_string.HTML:
            if index == 0:
                print( "The selected fruit is: " , child.value)
                box_values[ 'fruit' ] = child, value
            elif index == 1:
                print( "The select number of fruits is: " , str(child.value))
                box__values[ 'count' ] = child.value
            elif index == 2:
                print( "The selected type of fruit is: " , str(child.value))
                box_values[ 'type' ] = child.value
box values
```

UNCLASSIFIED //FOR OFFICIAL USE ONLY

Module: GUI Basics with Tkinter

Updated almost 2 years ago by [DELETED] in COMP 3321 (U) Module: GUI Basics with Tkinter

UNCLASSIFIED //FOR OFFICIAL USE ONLY

(U) Tkinter

(U) Tkinter comes as part of Python, so is readily available for use-just import it. Note: While Tkinter is almost always available, Python can be installed without it, e.g. if Tcl/Tk is not available when Python is compiled. Tk is a widget library that was originally designed for the Tcl scripting language, but now has been ported to Perl, Ruby, Python, C++ and more.

(U) NOTE: In Python 2 it must be used as Tkinter with a capital T.

(U) Setup

(U//FOUO) This lesson cannot currently be run from Jupyter on LABBENCH due to displayback limitations. It should work using Anaconda's Jupyter locally [DELETED]. The examples can also be copied to files and run as scripts from MACHINESHOP if your display is properly configured.

(U//FOUO) On MACHINESHOP

(U//FOUO) To display Python Tk objects back from MACHINESHOP, you will need to run the following on your MASH instance [DELETED]. NOTE: Not all steps may be necessary; verification needed.

```
yum -y groupinstall desktop
yum -y install tigervnc-server
yum -y install xrdp
/sbin/service xrdp start
chkconfig xrdp on
/sbin/service iptables stop
```

(U) What's a GUI (Graphical User Interface)?

(U) We all use them, some of us love them and hate them. Do we consider it 2- or 3-dimensional (2.5-dimensional)? Let's look at some very basic examples.

(U) Example 1

```
import tkinter as tk
root = tk.Tk()
root.mainloop()
```

We just created our first gui! But it doesn't do a whole lot yet. That is because we only created a blank/empty window that is waiting for our creation. tk.Tk( ) is the top level window that we will create for every gui that we make.

Parts of a gui: Choose widgets ==> Arrange in window ==> Add functionality

(U) Example 2

A first look at widgets!

```
#Basic gui with a Label, and a button

import tkinter as tk
root = tk.Tk()

label = tk.Label(root, text= "I am a label widget" ) # Create label
button = tk.Button(root, text= "I am a button" ) # create button

label.pack() # Add label to gui
button.pack() # Add button to gui
root.mainloop()
```

(U) Widget Types

Type	Description
Button	Users click on buttons to trigger some action. Button clicks can be translated into actions taken by your program. Buttons usually display text but can show graphics.
Canvas	A surface on which you can draw graphs and/or plots and also use as the basis of your own widgets.
CheckButton	A special type of Button that has two states; clicking changes the state of the button from one to the other.
Entry	Used to enter single lines of text and all kinds of input.
Frame	A container for other widgets. One can set the border and background color and place other widgets in it.
Label	Used to display pieces of text or images, usually ones that won't change during the execution of the application.
Listbox	Used to display a set of choices. The user can select a single item or multiple items from the list. The Listbox can be also rendered as a set of radio buttons or checkboxes.
Message	Similar to Text but can automatically wrap text to a particular width and height.
Menu	Used to put a menu in your window if you need it. It corresponds to the menu bar at the top but can also be used as a pop-up.
Menubutton	Adds choises to your Menus

Type	Description
Radiobutton	Represents one of a set of mutually exclusive choices. Selecting one Radiobutton from a set deselects any others.
Scale	Lets the user set numeric values by dragging a slider.
Scrollbar	Implements scrolling on a larger widget such as a Canvas, Listbox, or Text.
Text	A multi-line formatted text widget that allows the textual content to be "rich." It may also contain embedded images and Frames.
Toplevel	A special kind of Frame that interacts directly with the window manager. Toplevels will usually have a title bar and features to interact with the window manager. The windows you see on your screen are mostly Toplevel windows, and your application can create additional Toplevel windows if it is set to do that.

Other widgets: OptionMenu, LabelFrame, PanedWindow, Bitmap Class, Spinbox, Image Class

(U) Example 3

Let's look at some other widget examples. Also notice, that widgets have their own special "widget variables." Instead of using builtin python types, tk widgets use their own objects for storing this internal information. The tk widget variables are: StringVar, Intvar, DoubleVar, and BooleanVar

```
import tkinter as tk

root * tk.Tk()

tk.Label(root, text= "Enter your Password: " ).pack()
tk.Button(root, text= "Search" ).pack()

v = tk.IntVar()
tk.Checkbutton(root, text= " Remember Me" , variables). pack()
tk.Entry(root, width=30)

v2 = tk.IntVar()
tk.Radiobutton(root, text= "Male" , variable=v2, value=1).pack()
tk.Radiobutton(root, text= "Female" , variable=v2, value=2).pack()

var = tk.IntVar()
tk.OptionMenu(root, var, "Select Country" , "USA" , "UK" , "India" , "Others" ).pack()
tk.Scrollbar(root, orient= 'vertical' ).pack()

root.mainloop()
```

(U) Three ways to configure a widget

1. (U) Setting the values during initialization. (The way we have been doing it so far).
2. (U) Using keys to set the values.
3. (U) Using the widget's configure method.

(U) Widget Attributes

(U) There are tons of options that can be set, but here are. few of the important ones.

Attribute	Description
background / bg	The color of the body of the widget.e.g. 'red', 'blue', 'green', 'black').
foreground / fg	The color used for text.
padx, pady	The amount of padding to put around the widget horizontally and vertically. Without these the widget will be just big enough for its content.
borderwidth	Creates a visible border around a widget
height, width	Specifies the height and width of the widget.
disableforeground	When a widget is disabled, this is the color of its text (usually gray).
State	Default is 'normal' but also can use 'disabled' or 'active'

(U) Example 4

Let's try redoing Example 2 using the key/value method to make our label and the .configure()

method to make our button.

```
#Basic gui with a label and a button
import tkinter as tk
root = tk.Tk()

label = tk.Label(root)
label[ "text" ]= "I am a label widget" # using keys
button = tk.Button(root)
button.configure(text= "I am a button" ) # using configure

label.pack() #Add label to gui
button.pack() #Add button to gui
root.mainloop()
```

(U) Geometry Managers

(U) Now that we know how to create widgets, we're on to step two: arranging them in our window!

(U) There are mainly two types of geometry managers:

Pack

Grid (U) There is also a third--Place--but maybe that's not for today. (U) Quick, easy, effective.

If things get complicated, use Grid instead. | Attribute | Description | | ---------- |

-------- | | fill | Can be X, Y, or Both. X does the horizontal, Y does the vertical. | | expand | False

means the widget is never resized, True means the widget is resized when the container is

resized. | | side | Which side the widget will be packed against (TOP, BOTTOM, RIGHT or

LEFT). |

(U) Example 5

The pack geometry manager arranges widgets relative to window/frame you are putting them in.

For example, if you select side=LEFT it will pack you widget against the left side of the widget.

```
Example using pack geomtry manager
from tkinter import *
root = Tk()
parent = Frame(root)
# placing widgets top-down
Button(parent, text= 'ALL IS WELL' ).pack(fill=X)
Button(parent, text- 'BACK TO BASICS' ).pack(fill=X)
Button(parent, text= 'CATCH ME IFU CAN' ).pack(fill=X)
# placing widgets side by side
Button(parent, text= 'LEFT' ).pack(side=LEFT)
Button(parent, text= 'CENTER' ).pack(side=LEFT)
Button(parent, text= 'RIGHT' ).pack(side=LEFT)
parent.pack()
root.mainloop()
```

Example 6

Generally, the pack geometry manager is best for simple gui's, but one way to make more

complicated gui's using pack is to group widgets together in a Frame and then add the Frame to

your window.

```
#Example using pack geometry manager
from tkinter import *
root = Tk()
frame = Frame(root) #Add frame for grouping widgits
# demo of side and fill options
```

```
Label(frame, text=          "Pack Demo of side and fill"                ).pack()
Button(frame, text=           "A"   ).pack(side=LEFT, fill=Y)
Button(frame, text=           "B"   ).pack(side=TOP, fill=X)
Button(frame, text=           "C"   ).pack(side=RIGHT, fill=NONE)
Button(frame, text=           "D"   ).pack(side=TOP, fill=BOTH)
frame.pack()
# note the top frame does not expand nor does it fill in

# X or Y directions
# demo of expand options - best understood by expanding the root widget and seeing the effect on all the three
Label(root, text=          "Pack Demo of expand"          ). pack()
Button(root, text=           "I do not expand"          ).pack()
Button(root, text=           "I do not fill x but I do not expand"                ).pack(expand=1)
Button(root, text=           "I fill x and expand"          ). pack(fill=X, expands)
root.mainloop()
```

(U) Grid

(U) Use it when things get complicated, but it can be complicated in itself!

Attribute	Description
row	The row in which the widget should appear,
column	The column in which the widget should appear.
sticky	Can be., s, E, or. . One needs to actually see this work, but it's important if you want the widget to resize with everything else,
rowspan, columnspan	Widgets can start in one widget and occupy more than one row or column.

(U) Rowconfigure and columnconfigure

(U) Most layout definitions start with these. | Option | Description | | minsize | Defines the row's or column's minimum size. | | pad | Sets the size of the row or column by adding the specified amount of padding to the height of the row or the width of the column. | | weight | Determines how additional space is distrbuted between the rows and columns as the frame expands. The higher the weight, the more of the additional space is taken up. A row weight of. will expand twice as fast as that of 1. |

(U) Example 7

The grid geometry manager starts with row zero and column zero up in the top left hand corner of your window. When you add a widget using grid, the default is row=0 and column=0 so you don't

have to explicitly state it, although it is good practice.

```
#Basic exampLe using grid geometry manager
from tkinter import *
root = Tk()
Label(root, text= "Username" ).grid(row=0, sticky^W)
Label(root, text= "Password" ) .grid(row=l, sticky.)
Entry(root).grid(row=0, column=1, sticky=E)
Entry(root).grid(row=1, column=1, sticky=E)
Button(root, text= "Login" ). grid(row=2, column=1, sticky=E)
root.mainloop()
```

(U) Example 8

You could create this example using pack… But it you would probably need a lot of frames. Using grid for something like this is much easier!

```
#More advanced example using geometry manager
from tkinter import *
parent = Tk()
parent.title( 'Find & Replace' ) #Title of window

#First label and text entry widgits
Label(parent, text= "Find:" ) grid(row=0, column=0, sticky= 'e' )
Entry(parent, width=60).grid(row=0, column=1, padx=2, pady=2, sticky= 'we' , columnspan=9)

#second label and text entry widgits
Label(parent, text= "Replace:" ).grid(row=1, column=0, sticky= 'e' )
Entry(parent).grid(row=1, column=1, padx=2, pady=2, sticky= 'we' , columnspan=9)

#buttons
Button(parent, text= "Find" ) grid(
    row=0, column=10, sticky= 'e' + 'w' , padx=2, pady=2)
Button(parent, text= "Find All" ).grid(
    row=l, column=10, sticky= 'e' + 'w' , padx=2)
Button(parent, text= "Replace" ).grid(row=2, column=10, sticky= 'e' + 'w' , padx=2)
Button.parent, text= "Replace All" ).grid(
    row=3, column=10, sticky= 'e' + 'w' , padx=2)

#Checkboxes
Checkbutton(parent, text= 'Match whole word only' ).grid(
    row=2, column=1, columnspan=4, sticky= 'w' )
Checkbutton(parent, text= 'Match Case' ).grid(
    row=3, column=1, columnspan=4, sticky= 'w' )
Checkbutton(parent, text= 'Wrap around ' ).grid(
    row=4, column=1, columnspan=4, sticky= 'w' )

#Label and radio buttons
```

```
Label(parent, text=          "Direction: "          ).grid(row=2, column=6, sticky=          'w'     )
Radiobutton(parent, text=          'Up'      , value=1).grid(
        row=3, column=6, columnspan=6, sticky=          'w'     )
Radiobutton(parent, text=          'Down'       , value=2).grid(
        row=3, column=7, columnspan=2, sticky=          'e'     )

#run gui
parent mainloop()
```

(U) Object Oriented GUI's

Up until now, all our examples show how to use tkinter with out creating classes, but in reality, GUI programs usually get very large very quickly. To keep things organized, it's best to encapsulate your GUI in. class and group your widget creation inside of class functions.

(U) Example 9

Wrapping a GUI inside a class. Note that in real life, you probably won't be running. GUI from inside Jupyter. This example shows how to check if you are in main, which you will need if you are running from the command line.

```
from      tkinter          import          *
class          Application              (Frame):
        '''
        class Application :: Basic Tkinter example
        '''
        def      create_widgets              (self ):
                #Create widget
                self.hi_there = Button(self)
                self.hi_there[          'text'          ] =       'hello'
                self.hi_there[          'fg'       ] =       'blue'
                #Arrange widget
                self.hi_there.pack({          'side'          :     'left'          })

                #Create Midget
                self.QUIT = Button(self)
                self.QUIT[          'text'          ] =       'Quit'
                self.QUIT[          'fg'       ] =       'red'

                #Arrange widget
                self.QUIT.pack({          'side'          :     'left'          })

        def      __init__          (self, master = None):
                '''
                Constructor
                '''
                Frame.__init__( self.master)
```

```
        self.pack()
        self.create_widgets()

    def main ():
        root = Tk()
        app = Application(master=root)
        app.mainloop()

if __name__ == '__main__' :
    main()
```

(U) Callbacks and Eventing

(U)Most widgets have a command attribute to associate a callback function for when the widget is clicked. When you need your gui to respond to something other than a mouse click or a specific kind of mouse click, you can bind you widget to an event.

(U) Example 9.1

Add a simple callback function to example 9 using. What does the print command do when you click the "hello" button? What happens when you click the "Quit" button? Hint: you really are "quitting" your program. It just doesn't destroy (that is, close) your window!

```
from tkinter import *
class Application(Frame):
    '''
    class Application :: Basic Tkinter example
    '''

    #Callback function
    def say_hello(self) :
        print('Hello There')

    def create_widgets(self):
    #Create widget
    self.hi_there = Button(self)
    self.hi_there['text'] = 'hello'
    self.hi_there['fg'] = 'blue'

    #Arrange widget
    self.hi_there.pack ({'side': 'left'})
    #call back functionality
    self.hi_there['command'] = self.say_hello

    #Create widget
    self.QUIT = Button(self)
    self.QUIT['text'] = 'Quit'
```

```
        self.QUIT['fg'] = 'red'
        #Arrange widget
        self.QUIT.pack({'side': 'left'})
        # call back functionality
        self.QUIT['command'] = self.quit

        def __init__.self, master = None):
        """
        Constructor
        """

        Frame.__init__(self.master)
        self.pack()
        self.create_widgets()

        def main():
                root = Tk() #create window
                app Application(master=root)
                app mainloop()
if __name__ == '__main__':
        main()
```

(U) Example 9.2

(U) Using the key/value method made it easy to check we were following the pattern:

1. Choose widget
2. Add to window/arrange
3. Add functionality

But it is shorter to write the code using the initialize method. Even if your code isn't written in the order of these steps, this is still the order you want to think about them,

```
from tkinter import *

class Application(Frame):
        '''
        class Application :: Basic Tkinter example
        '''
        def say_hello(self):
                print('Hello There')
        def create_widgets(self):
                self.hi_there = Button(self, text='hello', fg='blue',command=self.say_hello)
                self.hi_there.pack(side='left')
                self.QUIT = Button(self,text ='quit', fg='red', command=self.quit)
                self.QUIT.pack(side='left')

        def __init__.self, master = None):
```

```
        '''
        Constructor
        '''
        Frame.__init__(self.master)
        self.pack()
        self.create_widgets()

    def main():
        root = Tk()
        app = Application(master=root)
        app.mainloop()

if __name__ == '__main__':
    main()
```

(U) Example 10

(U) Now that we have covered the three basics of GUI's: selecting widgets, arranging them in our window, and adding functionality, let's try a more complicated example.

```
from    tkinter    import    *
class    Application    (Frame):
    '''
    Application -- the main app for the Frame… It all happens here.
    '''
    # Let's define some Class attributes
    mainwin_rows=11
    mainwin_cols=15
    ALL=N+S+E+W

    def    __init__    (self, master=None):
        '''
        Constructor
        '''
        # Call Frames Constructor
        Frame.__init__(self, master)
        # Call Private Grid Layout method
        self._conf_self_grid_size()
        #---------------------------------

        #-- Make. checkerboard with LabeLs of different coLors
        self.checkers_main_win()

    def    __conf_self_grid_size    (self):
        '''
        I'm laying out the master grid
        '''
        # These next two Lines ensure that the grid takes up the entire
        # window
```

```
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0,weight=1)
        #---------------------------------------

        #-- Now creating a grid on the main window
        for i in range (self.mainwin_rows):
            self.rowconfigure(i, weight=1)
        for i in range(self.mainwin_cols):
            self.columnconfigure(j, weight=1)
        self.grid(sticky=self.ALL)
        #---------------------------------------

    def colorgen (self):
        '''
        Generator function that alternates between red and blue
        '''
        while True :
            yield 'red'
            yield 'blue'
    def checkers_main_win (self):
        '''
        Creates. checkerboard pattern grid layout
        '''
        colors = self.colorgen()
        for r in range(self.mainwin_rows):
            for c in range(self.mainwin_cols):
                txt = 'Item {0}, {1}' .format(r,c)
                l = Label(self.text=txt,bg=next(colors))
                l.grid(row=r,column=c,sticky=self.ALL)

    def main ():
        root=Tk()
        #set the size of our window
        root.geometry( '800x600' )
        #Add a title to our window
        root.title( 'Awesome Gui -- It is way COOL' )
        app = Application(master=root)
        app.mainloop()

if __name__ == '__main__' :
    main()
```

(U) Exercise 10.1

(U) We have a lovely grid with labels. We don't want to disturb our grid, so let's put. new Frame on top. Notice how our frame lines up on the grid. It makes it really easy to see how setting the row and column alignment works and also the rowspan and columnspan.

```
from tkinter import *
class Application (Frame):
    '''
    Application -- the main app for the Frame… It all happens here.
    '''
    # Let's define some CLass attributes
    mainwin_rows=11
    mainwin_cols=15
    ALL=N+S+E+W

    def __init__ (self, master=None):
        '''
        Constructor
        '''
        # Call Frames Constructor
        Frame.__init__(self, master)
        n -- Call Private Grid Layout method
        self._conf_self_grid_size()

        # Make a checkerboard with labels of different colors
        self.checkers_main_win()

        # Add Frame 1
        self.add_frame1()

    def conf_self_grid_size (self):
        """
        I'm laying out the master grid
        """
        # These next two lines ensure that the grid takes up the entire window
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0,weight=1)

        # Now creating a grid on the main window
        for i in range(self.mainwin_rows):
            self.rowconfigure(i, weight=1)
        for j in range(self.mainwin_cols):
            self.columnconfigure(j, weight=1)
        self.grid(sticky=self.ALL)

    def colorgen (self):
    '''
    Generator function that alternates between red and blue
    '''
    while True :
        yield 'red'
        yield 'blue'

    def checkers_main_win (self):
        """
        Creates a checkerboard pattern grid layout
```

```
        """
        colors = self.colorgen()
        for r in range(self.mainwin_rows) :
            for c in range(self.mainwin_cols) :
                txt = 'Item {0}, {1}' .format(r,c)
                l = Label(self,text=txt,bg=next(colors))
                l.grid(row=r,column=c, sticky=self.ALL)

    def add_frame1 (self):
        """
        Add a frame with a text area to put stuff in.
        """
        self.frame1=Frame(self,bg= 'red' )
        self.frame1.grid(row=0, column=0, rowspan=5, columnspan=6, sticky=self.ALL)

def main ():
    root=Tk()
    root.geometry( '800x600' )
    root.title( 'Awesome Gui -- It is way COOL' )
    app = Application(master=root)
    app.mainloop()

if __name__ == '__main__' :
    main()
```

(U) Exercise 10.2

Great! Now let's put a widget in our frame. Notice that while you can't mix grid and pack inside a conatainer, we can use pack inside our Frame even though we were using grid for our top level window. Exercise for reader: our Frame was red, but now that we added. text widget it doesn't look red anymore. What happened!? If we actually wanted. red text widget, what should we do differently?

```
from tkinter import *

class Application (Frame):
    """
    Application -- the main app for the Frame… It all happens here.
    """
    # Let's define some Class attributes
    mainwin_rows=11
    mainwin_cols=15
    ALL=N+S+E+W

    def __init__ (self,master=None):
        '''
        Constructor
        '''
```

```
        # Call Frames Constructor
        Frame.__init__(self,master)
        # -- Call Private Grid Layout method
        self.__conf_self_grid_size()
        # — Make a checkerboard with labels of different colors
        self.checkers_main_win()
        # - Add Frame 1 -
        self.add_frame1()

    def conf_self_grid_size (self):
        """
        I'm laying out the master grid
        """
        # These next two lines ensure that the grid takes up the entire window
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0, weight=1)
        # — Now creating a grid on the main window
        for i in range(self.mainwin_rows):
            self.rowconfigure(i, weight=1)
        for j in range(selfmainwin_cols):
            self.columnconfigure(j, weight=1)
        self.grid(sticky=self.ALL)

    def colorgen (self):
        """
        Generator function that alternates between red and blue
        """
        while True :
            yield 'red'
            yield 'blue'

def checkers_main_win (self):
    """
    Creates a checkerboard pattern grid layout
    """
    colors = self.colorgen()
    for r in range(self.mainwin_rows) :
        for c in range (self.mainwin_cols):
            txt = 'Item {0}, {1}' .format(r,c)
            l = Label(self.text=txt,bg=next(colors))
            l.grid(row=r, column=c, sticky=self.ALL)

    def add_frame1 (self):
        '''
        A frame is a nice way to show how to map out a grid
        '''
        self.frame1=Frame(self,bg= 'red' )
        self.frame1.grid(row=0, column=0, rowspan=5, columnspan=6, sticky=self.ALL)
        self.frame1.text_w=Text(self.frame1)
        self.frame1.text_w.pack(expand= True ,fill=BOTH)

def main ():
```

```
        root=Tk()
        root.geometry(          '800x600'       )
        root.title(             'Awesome Gui -- It is way COOL'                 )
        app = Application(master=root)
        app.mainloop()

if  __name__ ==         '__main__'      :
        main()
```

(U) Exercise 10.3

Let's add some another frame.

```
from    tkinter         import      *
class       Application         (Frame):
        """
        Application -- the main app for the Frame… It all happens here.
        """
        # Let's define some Class attributes
        mainwin_rows=11
        mainwin_cols=15
        ALL=N+S+E+W
        def     __init__        (self,master=None):
                '''
                Constructor
                '''
                # Call Frames Constructor
                Frame.__init__(self,master)
                # -- Call Private Grid Layout method
                self.__conf_self_grid_size()
                # — Make a checkerboard with labels of different colors
                self.checkers_main_win()
                # - Add Frame 1 -
                self.add_frame1()
                # - Add Frame 2 -
                self.add_frame2()

        def     conf_self_grid_size             (self ):
                """
                I'm laying out the master grid
                """
                # These next two Lines ensure that the grid takes up the entire window
                self.master.rowconfigure(0, weight=1)
                self.master.columnconfigure(0, weight=1)
                # — Now creating a grid on the main window
                for     i   in      range(self.mainwin_rows):
                        self.rowconfigure(i, weight=1)
                for     j   in      range(selfmainwin_cols):
                        self.columnconfigure(j, weight=1)
```

```
        self.grid(sticky=self.ALL)

    def colorgen(self):
        """
        Generator function that alternates between red and blue
        """
        while True:
            yield 'red'
            yield 'blue'

    def checkers_main_win(self):
        """
        Creates a checkerboard pattern grid layout
        """
        colors = self.colorgen()
        for r in range (self.mainwin_rows):
            for c in range(self.mainwin_cols):
                txt = 'Item{0}, {1}'.format(r,c)
                l = Label(self,text=txt,bg=next(colors))
                l.grid(row=r,column=c,sticky=self.ALL)

    def add_frame1(self) :
        """
        Add a frame with a text area to put stuff in.
        """
        self.frame1=Frame(self,bg='red')
        self.frame1.grid(row=0, column=0, rowspan=5, columnspan=6,sticky=self.ALL)
        self.frame1.text_w=Text(self.frame1)
        self.frame1.text_w.pack(expand=True,fill=BOTH)

    def add_frame2(self ):
        # Green frame
        self.frame2 = Frame(self,bg='green')
        self.frame2.grid(row=0, column=0, rowspan=5, columnspan=6,sticky=self.ALL)

def main():
    root=Tk()
    root.geometry('800x600')
    root.title('Awesome Gui -- It is way COOL')
    app = Application(master=root)
    app.mainloop()

if __name__ == '__main__':
    main()
```

What if we wanted our GUI to do something when we click on our green frame? Frame doesn't allow us to set command. But wait! Hope is not lost! We can bind our frame to an event…

Binding to an event

(U) Not all widgets have a command option, but that doesn't mean you can't interact with them. Also, there may be instances when you want a response from the user other than a mouse click (which is what the built in command function responds to). In these instances, you want to bind your widget to the appropriate event. Format: modifier(optional) - event type - detail(optional) For example: <Button-1> modifier is none, event type is Button and detail is one. This means the event is the left mouse button was clicked. For the right mouse button, you would use two as your detail. Common event types: Button, ButtonRelease, KeyRelease, Keypress, Focusln, FocusOut, Leave (when the mouse leaves the widget), and MouseWheel. Common modifiers: Alt, Any (used like <Any-KeyPress>), Control, Double (used like <Double-Button-1>) Common details: These will vary widely based on the event type. Most commonly you will specify the key for Keypress, ex: <KeyPress-F1>

Exercise 10.4

Let's bind our green frame to the left mouse button click and print out the coordinates of the click. Since printing to our notebook or the command line is not terribly useful for GUI's let's also display the coordinates in our text field

```
from tkinter import *
class Application (Frame):
    """
    Application -- the main app for the Frame… It all happens here.
    """
    # Let's define some Class attributes
    mainwin_rows=11
    mainwin_cols=15
    ALL=N+S+E+W

    def __init__ (self,master=None):
        '''
        Constructor
        '''
        # Call Frames Constructor
        Frame.__init__(self,master)
        # -- Call Private Grid Layout method
        self.__conf_self_grid_size()
        # — Make a checkerboard with labels of different colors
        self.checkers_main_win()
        # - Add Frame 1 -
        self.add_frame1()
        # - Add Buttons -
        self.add_frame2()

    def __conf_self_grid_size (self):
        """
        I'm laying out the master grid
        """
```

```
        # These next two lines ensure that the grid takes up the entire window
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0,weight=1)

        #--- Now creating a grid on the main window
        for     i   in      range(self.mainwin_rows):
                self.rowconfigure(i, weight=1)
        for     j   in      range(selfmainwin_cols):
                self.columnconfigure(j, weight=1)
        self.grid(sticky=self.ALL)

def     colorgen        (self):
    """
    Generator function that alternates between red and blue
    """
    while       True    :
            yield       'red'
            yield       'blue'

def     checkers_main_win               (self ):
        """
        Creates a checkerboard pattern grid layout
        """
        colors = self.colorgen()
        for     r   in      range (self.mainwin_rows):
                for     c   in      range(self.mainwin_cols):
                        txt =       'Item{0}, {1}'                  .format(r,c)
                        l = Label(self,text=txt,bg=next(colors))
                        l.grid(row=r,column=c,sticky=self.ALL)

def     add_frame1          (self) :
        """
        Add a frame with a text area to put stuff in.
        """
        self.frame1=Frame(self,bg=                          'red'       )
        self.frame1.grid(row=0,column=6,rowspan=10,columnspan=10,stick=self.ALL)
        self.frame1.text_w=Text(self.frame1)
        self.frame1.text_w.pack(expand=                         True    ,fill=BOTH)

def     add_frame2          (self):
        # Green frame!
        self.frame2 Frame(self, bg=                         'green'         )
        self.frame2.grid(row=0, column=0, rowspan=5, columnspan=6,sticky=self.ALL)
        self.frame2.bind(                   '<Button-1>'            ,self.frame2_handler)

def     frame2_handler              (self, event):
        """
        Handles events from frame2
        """
        msg =       'Frame 2 clicked at {} {}'                      .format(event.x, event.y)
        print(msg)
        self.framel.text_w.delete(1.0, END)
```

```
            self.framel.text_w.insert(END, msg)

def main ():
    root=Tk()
    root.geometry( '800x600' )
    root.title( 'Awesome Gui -- It is way COOL' )
    app = Application(master=root)
    app.mainloop()

if __name__ == '__main__' :
    main()
```

(U) Example 10.5

Great! Let's add some buttons now. Even though buttons have a command attribute, it can be tricky to pass information about the button being clicked using it. We would have to write a separate function for each button! Instead, let's use a key binding so we can access the button text from the event. Notice when we arrange the buttons we have to allign them by incriments of three since they span three columns.

```
from tkinter import *
class Application (Frame):
    """
    Application -- the main app for the Frame… It all happens here.
    """
    # Let's define some Class attributes
    mainwin_rows=11
    mainwin_cols=15
    ALL=N+S+E+W

    def __init__ (self,master=None):
        '''
        Constructor
        '''
        # Call Frames Constructor
        Frame.__init__(self,master)
        # -- Call Private Grid Layout method
        self.__conf_self_grid_size()
        # — Make a checkerboard with labels of different colors
        self.checkers_main_win()
        # - Add Frame 1 -
        self.add_frame1()
        # - Add Buttons -
        self.add_buttons()

    def __conf_self_grid_size (self):
        """
        I'm laying out the master grid
```

```
        """
        # These next two lines ensure that the grid takes up the entire window
        self.master.rowconfigure(0, weight=1)
        self.master.columnconfigure(0,weight=1)

        #--- Now creating a grid on the main window
        for i in range(self.mainwin_rows):
            self.rowconfigure(i, weight=1)
        for j in range(selfmainwin_cols):
            self.columnconfigure(j, weight=1)
        self.grid(sticky=self.ALL)

    def colorgen (self):
        """
        Generator function that alternates between red and blue
        """
        while True :
            yield 'red'
            yield 'blue'

    def checkers_main_win (self):
        """
        Creates a checkerboard pattern grid layout
        """
        colors = self.colorgen()
        for r in range (self.mainwin_rows):
            for c in range(self.mainwin_cols):
                txt = 'Item{0}, {1}' .format(r,c)
                l = Label(self,text=txt,bg=next(colors))
                l.grid(row=r,column=c,sticky=self.ALL)

    def add_frame1 (self) :
        """
        Add a frame with a text area to put stuff in.
        """
        self.frame1=Frame(self,bg= 'red' )
        self.frame1.grid(row=0,column=6,rowspan=10,columnspan=10,stick=self.ALL)
        self.frame1.text_w=Text(self.frame1)
        self.frame1.text_w.pack(expand= True ,fill=BOTH)

    def add_frame2 (self):
        # Green frame!
        self.frame2 Frame(self, bg= 'green' )
        self.frame2.grid(row=0, column=0, rowspan=5, columnspan=6,sticky=self.ALL)
        self.frame2.bind( '<Button-1>' ,self.frame2_handler)

    def frame2_handler (self, event):
        """
        Handles events from frame2
        """
        msg = 'Frame 2 clicked at {} {}' .format(event.x, event.y)
        print(msg)
```

```
                self.framel.text_w.delete(1.0, END)
                self.framel.text_w.insert(END, msg)

        def     add_buttons             (self ):
                """
                Add buttons to the bottom
                """
                self.button_list=[]
                button_labels [                 'Red'     ,    'Blue'      ,    'Green'       ,    'Black'       ,    'yellow'          ]
                for     c,bt      in     enumerate(button_labels):
                        b = Button( self.text=bt)
                        #span three columns and set the column alignment as multiples of three.
                        b.grid(row=10, column=c*3, columnspan=3, sticky=self.ALL)
                        #Bind buttons to button click.
                        b.bind(       '<Button-1>'              , self.buttons_handler)
                        self.button_list.append(b)

        def     buttons_handler                 ( self, event):
                '''
                Event Handler for the buttons
                '''
                button_clicked = event widget[                          'text'        ]
                print(button_clicked)

                self.frame1.text_w.delete(1.0, END)
                self.frame1.text_w.insert(END, button_clicked)

def     main    ():
        root=Tk()
        root.geometry(              '800x600'         )
        root.title(             'Awesome Gui -- It is way COOL'                         )
        app = Application(master=root)
        app.mainloop()

if      __name__ ==             '__main__'          :
        main()
```

(U) Dialogs

(U) Tkinter provides a collection of ready-made dialog boxes to use:

- showinfo
- showwarning
- showerror
- askquestion
- askokcancel
- askyesno

askyesnocancel

askretry_cancel

```
from tkinter import messagebox as mBox
from tkinter import Tk

root = Tk()
#TMs keeps the top Level window from being drawn
root withdraw()

mBox.showinfo( 'Python Message Info Box' , 'A Python GUI created using tkinter:\nThe year is 2016.'
# mBox.showwarning('Python Message Morning Box', 'A Python GUI created using tkinter:\nWarning: There might be
# mBox.showerror('Python Message Error Box', 'A Python GUI created using tkinter:\nError: Houston ~ we DO have
```

Useful References:

Python GUI Programming Cookbook By Burkhard A. Meier On safari: http://ncmd-ebooks-1.ncmd.nsa.ic.gov/9781785283758 Great overview of different widgets and also using.themed" tk widgets or ttk widgets.

Tkinter GUI Application Development Blueprints By: Bhaskar Chaudhary On safari: https://ncmd-ebooks-1.ncmd.nsa.ic.gov/9781785889738

Not ail of the packages used in the examples are available, but this book is still. great reference for how to build and structure larger GUI projects.

Programming Python, 4th Edition By: Mark Lutz On safari: https://ncmd-ebooks-1.ncmd.nsa.in.gov/9781449398712

Not a dedicated GUI book, this book does have several chapters cover different GUI aspects. It spends more time explaining the underlying logic of GUI's and covering special cases that can trip you up. It starts from the basics, but moves quickly, so. might not be the best resource for total novices.

UNCLASSIFIED//FOR OFFICIAL USE ONLY

Python GUI Programming Cookbook

Updated over 1 year ago by [DELETED] (U) Code from "Python GUI Programming Cookbook"

Python GUI Programming Cookbook By Burkhard A. Meier On safari: http://ncmd-ebooks-1.ncmd.nsa.ic.gov/9781785283758

```
#=======
```

```
#Imports
#=======
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
from tkinter import Menu
from tkinter import Spinbox
from tkinter import messagebox as mBox
# basic gui
# "themed tk" improved gui options
# For scrolling text boxes
# For creating menus
# For creating spin boxes
# For message boxes
#=============================================================
# Initial framework setup for window^ tabs, frameSj etc.
#=============================================================
win = tk.Tk()                                                    # Create instance
win.title( "Python GUI" )                                        # Add a title
win.iconbitmap( r'U:\private\anaconda3\DLLs\pyc.ico' )            # Change icon

tabControl = ttk.Notebook(win)                                   # Create Tab Control

tab1 = ttk.Frame(tabControl)                                     # Create a tab
tabControl.add(tab1, text= 'Tab 1' )                             # Add the tab

tab2 = ttk.Frame(tabControl)                                     # Add a second tab
tabControl.add(tab2, text= 'Tab 2' )                             # Make second tab visible

tab3 = ttk.Frame(tabControl)                                     # Add a third tab
tabControl.add(tab3, text= 'Tab 3' )                             # Make second tab visible

tabControl.pack(expand=1, fill= "both" )                         # Pack to make visible

# We are creating a container frame to hold all other widgets in tab1
monty = ttk.LabelFrame(tab1, text= 'Monty Python' )
monty.grid(column=0, row=0, padx=8, pady=4)

# We are creating a container frame to hold all other widgets in tab2
monty2 = ttk.LabelFrame(tab2, text= 'The Snake' )
monty2.grid(column=0, row=O, padx=8, pady=4)

#====================
# Callback functions
#====================

def clickMe ():      # Function for when button is clicked
    action.configure(text= 'Hello' + name.get() + ' number ' , + numberChosen.get()+ '!' )

# Set Radiobutton global variables into a list.
colors = [ "Pink" , "Magenta" , "Purple" ]
# We have also changed the callback function to be zero-based, using the list instead of module-level global v
```

```
# Radiobutton callback function

def radCall():
    radSel=radVar.get()
    if radSel == 0: monty2.configure(text=colors[0])
    elif radSel == 1: monty2.configure(text=colors[l])
    elif radSel == 2: monty2.configure(text=colors[2])

def _quit():
    win.quit()
    win.destroy()
    #exit()

# Display a Message Box
# Callback function
def _msgBox():
    #mBox.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2016.')
    #mBox.showwarning('Python Message Warning Box', 'A Python GUI created using tkinter:\nWarning: There might
    #mBox.showerror('Python Message Error Box', 'A Python GUI created using tkinter:\nError: Houston ~ we DO h
    answer = mBox.askyesno( "Python Message Dual Choice Box" , "Are you sure you really wish to do this?"
    print(answer)

# Spinbox callback
def _spin():
    value = spin.get()
    print(value)
    scr.insert(tk.INSERT, value + '\n' )
#================
# Create Menu bar
#================
menuBar = Menu(win)                                                    # create menu bar
win.config(menu=menuBar)                                               # add menu bar to gui

fileMenu = Menu(menuBar, tearoff = 0)                                  #create menu
fileMenu. add_command(label= "New" )                                   #add option to menu
fileMenu.add_separator()                                               #add separator to menu
fileMenu.add_command(label= "Exit" , command=_quit)                    #add option to menu
menuBar.add_cascade(label= "File" , menu=fileMenu)                     #add menu to menu bar

# Add another Menu to the Menu Bar and an item
helpMenu = Menu(menuBar, tearoff=0)                                    #create second menu
helpMenu add_command(label= "About" , command=_msgBox)                 #add menu item
menuBar.add_cascade(label= "Help" , menu=helpMenu)                     #add menu to menu bar

#==============
# Contents Tab1
#==============
# Create Labels
ttk.Label(monty, text= "Enter a name:" ) .grid(column=0, row=0, sticky=tk.W)                    # text Label position.0,0)
ttk.Label(monty, text= "Choose a number:" ). grid(column=1, row=0, sticky=tk.W)                 # Another Label.with Location)

#===============
```

```
# Text entry box
#================
# Adding a Textbox Entry widget
name = tk.StringVar()                        # tk's version of a string (storage variable)
nameEntered = ttk.Entry(monty, width=12, textvariable=name)                                    # Entry box
nameEntered.grid(column=0, row=1, sticky=tk.W)                                  # Entry box Location
nameEntered.focus()                    # when app starts, put curser in box

#==========
# Combo box
#==========
number = tk.StringVar()                        # tk string variable to hold numbewr
numberChosen = ttk.Combobox(monty, width=12, textvariable=number, state =                                  'readonly'          )   #combo box
numberChosen[          'values'          ] = (1, 2, 4, 42, 100)                  # options for combo box
numberChosen.grid(column=1, row=1,sticky=tk.W)                                  # Location combo box
numberChosen.current(0)

#=======
# Button
#=======
action = ttk.Button(monty, text=                "Click Me!"          , command=clickMe)              # create button with text and function for when
action.grid(column=2, row=1, sticky=tk.W)                                  # Position Button in second row, second column.zero-based)

#========
# Spinbox
#========
# Adding a Spinbox widget
spin = Spinbox(monty, values=(1, 2, 4, 42, 100), width=5, bd=8, command=__spin)                                  #spinbox set of values
spin.grid(column=0, row=2)

# Adding a second Spinbox widget
spin2 = Spinbox(monty, from_=0, to=10, width=5, bd=8, relief = tk.RIDGE, command=_spin)                                  #spinbox range of
                                                                                                                         valu

#Alternate relief options:
#spin2 = Spinbox(monty, values=(0, 50, 100), width=5, bd=20, relief = tk.FLAT, command=_spin)
#spin2 = Spinbox(monty, values=(0, 50, 100), width=5, bd=20, relief = tk.GROOVE, command=_spin)

spin2.grid(column=1, row=2)

#===========
# Checkboxes
#===========
# Creating three checkbuttons
chVarDis = tk.IntVar()                        # tk int variable, for check box state
check1 = tk.Checkbutton(monty, text=                "Disabled"          , variable=chVarDis, state=                ' disabled'          )   # disabled checkbox
check1.select()                # add checkmark
check1.grid(column=0, row=4, sticky=tk.W)                                  # position checkbox. sticky=tk.W means aligned west
chVarlln = tk.IntVar()                        # another tk in variable for checkbox state
check2 = tk.Checkbutton(monty, text=                "UnChecked"          , variable=chVarlln)                  #un-checked checkbox
check2.deselect()                # set checkbox to not checked
check2.grid(column=1, row=4, sticky=tk.W)                                  # position checkbox
chVarEn = tk.IntVar()                        # checkbox int variabLe for checkbox state
```

```
check3 = tk.Checkbutton(monty, text=                    "Enabled"          , variable=chVarEn)                    # Checked checkbox
check3.select()                    # set checkbox to checked
check3.grid(column=2, row=4, sticky=tk.W)                                  # position checkbox

#==========
# Scrollbox
#==========
# Using a scrolled Text control
scrolW = 30                # scrollbox width
scrolH = 3                 # scrollbox height
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH, wrap=tk.WORD)                                                                    # create scroll box
scr.grid(column=0, row = 5, columnspan=3, sticky=                                  'WE'    )    # position scroll box

#+++++++++++++++
# Contents Tab2
#+++++++++++++++
#==============
# Radio Buttons
#==============

# create three Radiobuttons using one variabLe
radVar = tk.IntVar()

#Next we are selecting a non-existing index value for radVar.
radVar.set(99)

#Now we are creating all three Radiobutton widgets within one Loop.
for      col      in      range(3):
         curRad =          'rad'          + str(col)
         curRad = tk.Radiobutton(monty2, text=colors[col], variable=radVar, value=col, command=radCall)
         curRad.grid(column=col, row=6, sticky=tk.W)

#========================
# labels in a labelsframe
#========================
# Create a container to hold labels
labelsFrame = ttk.LabelFrame(monty2, text=                                  ' Labels in a Frame '                          )
labelsFrame.grid(column=0, row=7)                                  # position with padding

# Place labels into the container element
ttk.Label(labelsFrame, text=                          "Label1"           ).grid(column=0, row=0)
ttk.Label(labelsFrame, text=                          "Label2"         ) .grid(column=0, row=1)
ttk.Label(labelsFrame, text=                          "Label3"         ) .grid(column=0, row=2)

#+++++++++++++++
# Contents Tab3
#+++++++++++++++

tab3 = tk.Frame(tab3, bg=                        'purple'          )
tab3.pack()

#==================
```

```
#Callback functions
#==================

def checked():
    if chVarCr.get():
        canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='blue')
        canvas.grid(row=1, column=0)
        canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='blue')
        canvas.grid(row=0, column=1)
    else:
        canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='green')
        canvas.grid(row=1, column=0)
        canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='green')
        canvas.grid(row=0, column=1)

#======================
# Checkbox and canvases
#======================

for pinkColor in range(0,2):
    canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='pink')
    canvas.grid(row=pinkColor, column=pinkColor)

chVarCr = tk.IntVar()                    # another tk in variabie for checkbox state
checker = tk.Checkbutton(tab3, text="Color", variable=chVarCr. command=checked)    # un-checked checkbox
checker.deselect()                    # set checkbox to not checked
checker.grid(row=0,column=0)                    # position checkbox

# DispLay GUI
win.mainloop()
```

Examples of other message boxes

```
from tkinter import messagebox as mBox
from tkinter import Tk
root = Tk()
root.withdraw()
mBox.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2016.'
#mBox.showwarning('Python Message Warning Box'j.A Python GUI created using tkinter:\nWarning: There might be.
#mBox.showerror('Python Message Error Box ', 'A Python GUI created using tkinter:\nError: Houston ~ we DO ha\j


from tkinter import *
root = Tk()
root.withdraw()
messagebox.showerror( 'Python Message Error Box', 'A Python GUI created using tkinter:\nError: Houston ~ we DO

from tkinter import messagebox as mBox
```

```
help(mBox)
```

Module: Logging

Updated over 2 years ago by [DELETED] in COMP 3321 (U) Module: Logging

(U) There comes a time in every developer's career when he or she decides that there must be something better than debugging with print statements. Fortunately, it only takes about one minute to set up basic logging in Python, which can help you diagnose problems in your program during development, then suppress all that output when you use the program for real. After that, there's no looking back; you might even start to use a real debugger someday. The logging module has many advanced configuration options, but you'll probably see benefits even if you don't ever use any of them. (U) At its heart, logging is a name for the practice of capturing and storing data and events from a program that are not part of the main output stream. Logging also frequently uses the concept of severity levels: some captured data indicates serious problems, other data provides useful information, and some details are useful only when trying to track down bugs in the logic. Capturing messages from any or all of the differentseverity levels depends on who invoked the program, along with how and why it's being run.

(U) The Basics

(U) The logging module is included in the standard library. To begin using it with the absolute minimum effort possible, import it and start writing messages at different levels,

```
import logging
logging.warning('You have been warned')
logging.critical('ABORT ABORT ABORT')
logging.info('This is some helpful information')
```

(U) The logging module has several levels, including DEBUG, INFO, WARNING, ERROR, and CRITICAL, which are just integer constants defined in the module. Custom levels can bee added with the addLevelName method, but the default levels should usually be sufficient. Each of the module-level functions warning, critical, and info is shorthand for a collection of functionality, which can be accessed through separate methods if fine-grained control is needed.

Create a log message with severity at the level indicated (the method log(level, message) could also be used).

Send that message to the root logger

If the root logger is configured to accept messages of that level (or lower), send the message to the default handler, which formats the message and prints it to the console.

(U) This explains why the call to logging.info did not print to the screen: its severity is not high enough. By default, the root logger is set to only handle messages at the WARNING level or higher. The level of the root logger can be configured at the module level, but must be done before any messages are sent. Otherwise, the level must be set directly on the logger.

```
[(level, getattr(logging,level)) for level in [ 'DEBUG' , 'INFO' , 'WARNING' , 'ERROR' , 'CRITICAL'
logging.root.getEffectiveLevel()
logging.root.getEffectiveLevel() == logging.WARNING # True
logging.root.setLevel(logging.INFO)
logging.root.getEffectiveLevel() == logging.INFO # True
logging.info( "Now this should get logged." )
logging.log(21, "This will also get logged at a numbered custom level" )
logging.basicConfig(level=logging.DEBUG)
logging.root.getEffectiveLevel() == logging.DEBUG # False
exit()

## New Session
import logging
logging.basicConfig(level=logging.INFO)
logging.info ( "This is some information." )
logging.root getEffectiveLevel() == logging.INFO # True
```

(U) From all this, a strategy emerges around using logging to improve your debugging.

Instead of print statements, add calls to logging.debug to your code.

At the top of your script, use logging.basicConfig(level=logging.DEBUG) during development; switch to level=logging.INFO or level=logging.WARNING for production.

Optionally, make a command-line option for your script that enables debugging output (e.g. my_script.py --verbose). (U) So far, we have dealt with only the defaults: the logger logging.root, along with its associated Handler and Formatter. A program can create multiple loggers, handlers, and formatters and connect them all together in different ways. Aside from the StreamHandler already encountered, the FileHandler is also very common; it writes messages to. file named in its constructor. For ease of exposition, we use only StreamHandlers in our examples. Each handler has a setLevel method; a handler acts on a message only only if both the logger and the handler have levels set at or below the severity level of the message. Unless logging.basicConfig() has been called, handlers and formatters must be explicitly defined.

```
logging.basicConfig()
warnlog = logging.getLogger( 'warnings' )
warnlog.setLevel(logging.WARN)
infolog = logging.getLogger( 'info' )
infolog.setLevel(logging.INFO)
infolog.info( 'An informational message' )
info_handler = logging.StreamHandler()
```

```
infolog.addHandler(info_handler)
qmformatter = logging.Formatter('%(name)s???%(message)s???')
info_handler.setFormatter(qm_formatter)
info_handler.setLevel(logging.ERROR)
warnlog.warning('There it goes')
warnlog.info('Not there anymore')
infolog.info("It's coming back")
infolog.error("Oops, it didn't make it")
```

(U) In this example, infolog has two handlers: the default handler created by basicConfig and the explicitly set info_handler with its distinctive ??? -inspired formatter. This second handler only logs messages with severity equal to or higher than logging.ERROR, even though the infolog passes it messages with severity level as low as logging.INFO, as can be seen by the fact that the default handler prints these messages.

(U) Advanced Usage

(U) A variety of handlers have been written for different purposes. The RotatingFileHandler from the logging.handlers submodule automatically starts new log files when they reach. size, and keeps only. specified number of backups. The TimedRotatingFileHandler is similar, but time-based instead of size-based. Other handlers write messages to sockets, email, and over HTTP, TCP, or UDP. (U) Formatters use old-style string formatting with %, and have access to a dictionary which contains several interesting properties, including the name of the logger, the level of severity, the current time, and other information about the environment surrounding the logging message. Custom keys can be passed to the formatter with a dictionary passed via the extra keyword in logging.log or related shortcut methods, e.g. logging.warning. (U) If the configured levels are too restrictive, custom levels can be added. They must be assigned. number, which determines when messages at that level will be handled. No shortcut method is added for custom levels, so calls must be made to the log method. Of course, it's easy to add such a shortcut to the Logger class.

```
INFOWARN = 25
logging.addLevelName(INFOWARN, 'INFOWARN')
def infowarn(self, message):
self.log(INFOWARN, message)
logging.Logger.infowarn = infowarn
logger = logging.getLogger('info.warn')
logger.infowarn ("Halfway between info and warning.")
```

Module: Math and More

Updated about 2 years ago by [DELETED] in COMP 3321 (U) Module: Math and More

Math and More

The Math Module is extremely powerful… for doing math-y things. This page is mostly to demonstrate some of the neat things Python can do that are math related. We'll start with the math module. You can take this. step further and investigate cmath for complex operations. Most of what is in math is also in cmath.

math.py

```
import math
print(dir(math))
```

Constants of note

(Greek letter pi): Ratio of circle's circumference to its diameter,

```
math.pi
```

Euler's number, e, (pronounced "Oil-er"): The mathematical constant that is the base of the natural logarithm $$ e = \sum_{n=0}^{\infty} 1/n! = \lim_{n\to\infty} (1 + 1/n)^n $$ $$ e=\sum n = 0\infty1n!!=\lim n —> \infty(1+1n)n $$

```
# Euler's number.pronounced.Oil-er")
# The mathematical constant
math.e
```

Logs and Exponents

```
math.log10(10)
math.log10(100)
math.log10(1000)
math.log10(1)
math.log10(0.1)
math.log10(0)
math.log10(-1)
2**3
# buiit-in
pow(2,3)
math.pow(2,3)                # Converts arguments to fLoots
```

The following demonstrating modular exponentiation. Some ways are WAY faster than others…

```
pow(2,3,5)
(2**3) %5
pow(100,10000,7)
(100**10000) % 7
pow(100, 1000000, 7)
( 100 ** 1000000 ) % 7
```

Trig Functions

Notice argument to the trig function is measured in radians, not degrees.

```
help(math.sin)
math.sin(0)
```

If radians the following would be.:

```
math.sin(90)
```

So, we expect the following to be one:

```
math.sin(math.pi/2)
```

And the following should be. (halfway around the unit circle):

```
math.sin(math.pi)
```

Is this zero?!!? See the e-16 at the end? That's pretty close, say within rounding error, of 0. Just be careful that you don't check that it is exactly zero.

```
help(math.isclose)
print(math.sin(math.pi) == 0)
print(math. isclose(math.sin(math.pi), 0, abs_tol = 10**-15))
help(math.degrees)
help(math.radians)
math.degrees(math.pi)
math.radians(45) == math.pi / 4
```

Fun Example:

Save the first 1,000 of the Fibonacci sequence to a list (starting with 1,1 not 0,1). Iterate over that list and print out how many digits each number is.

```
def fib__list (n = 1000, init - [1,1]):
    '''Returns a list of the first n Fibonacci numbers.'''
    fib_list = init
    for x in range(n - 2):
        fib_list.append(fib_list[-1] + fib_list[-2])
    return fib_list

def fib_lengths (fib__list):
    '''Returns a list containing the number of digits in each fibonacci number.
    It can be calculated this way because the list passed in are integers.
    Use as intended!'''
    return [len(str(int(x))) for x in fib_list]

a = fib_list()
print(a[:20])

a_lengths - fib_lengths(a)
print(a_lengths[ :20])

b = fib_list(init = [1.0,1.0])
print(b[:20])

b_lengths = fib_lengths(b)
print(b_lengths[:20])

def fib_lengths (fib_list):
    '''Returns a list containing the number of digits in each fibonacci number.'''
    return [1 + math.floor(math.log10(x)) for x in fib_list]

b_lengths = fib_lengths(b)
print(b_lengths[:20])
```

NumPy

Pronounced "Num - Py"… not like stumpy… If running through labbench, run the next two lines. If running on jupyter-notebook through Anaconda, you can just import numpy.

```
import ipydeps

ipydeps.pip( 'numpy' )
```

Now we can import numpy:

```
import numpy as np
```

numpy's main object is called ndarray . It is:

homogeneous

multidimensional

array Meaning, it is a table of elements, usually numbers. All elements are the same type.

Elements are accessed by a tuple of positive integers. Dimensions are called axes. The number

of axes is the rank,

```
t = np.array([1,2,1])
t

np.ndim(t)          # used to be np.rank, but this is deprecated
a = np.array(np.arange(15).reshape(3,5))
a
np.ndim(a)

a.shape
a.ndim
a.dtype.name

# size in bytes … Like sizeof operator, but easier to get to.
a.itemsize
a.size
type(a)
```

We have three ways to access "row" 1 of a:

```
a[1] a[1,] a[1,:]
a[1]
```

To get column in position 2:

```
a[:,2]
```

To get single element "row" 0, column 2:

```
a[0,2]
```

Creating and modifying array:

```
c = np.array([2,3,4])
c.dtype.name
d = nparray([1.2,3.5,5.1])
d. dtype.name
```

Let's change one element of.:

```
c[1] = 4.5
c.dtype.name
```

Ut oh, the type didn't change. But we tried to add. float! Let's see what happened:

```
c
```

The array was updated, but the value we were adding was converted to an int64. Be careful. The type matters!! This doesn't work:

```
f = np.array(1,2,3,4)
```

Needs to be this:

```
f = np.array([1,2,3,4])
```

Interprets a sequence of sequences as a two dimensional array. Sequence of sequence of sequences as a three dimenional array… you get the pattern?

```
g - np.array([[1.5,2,3],[4,5,6]])
g
```

Type can be specified at creation time:

```
h = np.array([[1,2],[3,4]], dtype = complex)
print(h)
print()
print(h.dtype.name)
```

Remember c? Here is how we can change it from integers to floats:

```
print(c)
c = np.array(c, dtype = float)
```

```
print(c)
c[1] = 4.5
print(c)
```

Suppose you know what size you want, but you want it to have all zeros or ones and you don't want to write them all out. Notice there is one parameter we are passing for the dimensions, but it is a tuple.

```
np.zeros((3,4))
np.zeros((3,4), dtype = int)
np.ones((2,3 > 4), dtype = np.intl6)
```

The following is FAST, but dangerous. It does not initialize the entries in the array, but takes whatever is in memory. USE WITH CAUTION.

```
np.empty((2,3))
```

You know range? Well, there is a range. Which allows us to create an array containing a range of numbers evenly spaced.

```
np.arange(10. 30,5)                    # Start, stop, step
np.arange(0.2,0.3, 0.01)                        # Can do fLoots!!
```

Say we don't want to do the math to see what our step should be but we know how many numbers we want… that's what linspace is for! We get evenly spaced samples calculated over the interval. It takes a start, stop and the number of samples you want. There are other optional arguments, but you can figure those out!

```
np.linspace(0,2,9)                    # 9 numbers evenLy spaced numbers between 0 and 2, INCLUSIVE
```

Playing with "matrices"… or are they? The following are not necessarily the results of matrix operations. What exactly is going on here?

```
a
a + 1
a / 2
a // 2
pow(2,a)
b = a // 2
a + b
          # Not matrix muLtipLcation
a * b
b.T      # Transpose
```

Matplotlib

```
## Only if you are on Labbench. If using anaconda you don't need to do this .
## If you haven't already improt ipydeps, uncomment the next Line also before running
# import ipydeps
ipydeps.pip( 'matplotlib' )
import matplotlib.pyplot as pit
## If you haven't imported numpy, do so!
#import numpy as np
```

Line Plot

```
a = np.linspace(0,10,100)
b = np.exp(-a)
pit.plot(a,b)
pit.show()
```

Histogram

```
from numpy.random import normal, rand
x = normal(size = 200)
pit.hist(x,bins = 30)
pit.show()
```

3D Plot

```
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.gca(projection= '3d' )
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm)
pit.show()

plt.plot([1,2,3], [1,2,3], 'go-' , label= 'line 1' , linewidth=2)
plt.plot([1,2,3], [1,4,9], 'rs' , label= 'line 2' )
plt.axis([-2, 5, -2, 10])
```

```
plt.plot(a, b, color='green', linestyle='dashed', marker='o', markerfacecolor='blue', markersize=
plt show()
```

Sage

To be filled in…

COMP3321: Math, Visualization, and More!

Updated almost 3 years ago by [DELETED] in COMP 3321 (U) Python Math, Visualization, and More!

Python Math, Visualization, and More!

This notebook will give a very basic overview to some mathematical and scientific packages in Python, as well as some tools to visualize data.

Credit:

Much of this material is developed by Continuum Analytics, based on a tutorial created by R.R. Johansson.

What is NumPy?

Numpy is a Python library that provides multi-dimensional arrays, matrices, and fast operations on these data structures.

NumPy arrays have:

- fixed size
  - all elements have the same type -that type may be compound and/or user-defined
  - fast operations from:
    - vectorization — implicit looping
    - pre-compiled. code using high-quality libraries
      - NumPy default
      - BLAS/ATLAS
      - Intel's MKL

NumPy's Uses and Capabilities

- Image and signal processing

Linear algebra

Data transformation and query

Time series analysis

Statistical analysis

Numpy Ecosystem

```
#Run this if on LABBENCH
import ipydeps
packages = [ 'matplotlib ' , 'numpy' ]
for i in packages:
ipydeps.pip(i)
# Let's install necesary packages
import numpy as np
import matplotlib as mpl
import numpy.random as npr
vsep = "\n----------\n"
```

matplotlib — 2D and 3D plotting in Python

```
# This Line configures matplot Lib to show figures embedded in the notebook,
# instead of opening a new window for each figure. More about that later.
# If you are using an oLd version of IPython, try using '%pylab inline' instead.
%matplotlib inline
```

Introduction

Matplotlib is an excellent 2D and 3D graphics library for generating scientific figures. Some of the many advantages of this library include:

Easy to get started

Support for LaTeX formatted labels and texts

Great control of every element in. figure, including figure size and DPI.

High-quality output in many formats, including PNG, PDF, SVG, EPS, and PGF.

GUI for interactively exploring figures and support for headless generation of figure files.useful for batch jobs).

Matplotlib is well suited for generating figures for scientific publications because all aspects of the figure can be controlled programmatically. This is important for reproducibility, and convenient

when one needs to regenerate the figure with updated data or change its appearance.

More information at the Matplotlib web page: http://matplotlib.org/

To get started using Matplotlib in a Python program, import the matplotlib.pyplot module under the name plt:

```
import matplotlib.pyplot as plt
```

The matplotlib MATLAB-like API

A great way to get started with plotting using matplotlib is to use the MATLAB-like API provided by matplotlib. It is designed to be compatible with MATLAB's plotting functions, so if you are familiar with MATLAB, start here.

```
x = np.linspace(0, 5, 100)
y = x ** 2

x, y

plt.figure()
pit.plot(x, y, 'g' )
plt.xlabel( 'x' )
plt.ylabel( 'y' )
pit.title( 'title' )
pit.show()

pit.subplot(1,2,1)
pit.plot(x, y, 'r--' )
pit.subplot(1,2,2)
pit.plot(y, x, 'g*-' );
```

The matplotlib object-oriented API

The main idea with object-oriented programming is to have objects to which one can apply functions and actions, and no object or program states should be global.such as the MATLAB-like API). The real advantage of this approach becomes apparent when more than one figure is created, or when a figure contains more than one subplot. To use the object-oriented API, we start out very much like in the previous example, but instead of creating a new global figure instance, we store a reference to the newly created figure instance in the fig variable, and from it we create a new axis instance axes using the add_axes method in the Figure class instance fig:

```
fig plt.figure()
graph = fig.add_axes([0, 0, 1, 0.3])  # Left, bottom, width, height.range. to.)
```

```
graph.plot(x, y, V)
graph.set_xlabel( 'x' )
graph.set_ylabel( 'y' )
graph.set_title( 'title' );
```

Ithough a bit more code is involved, the advantage is that we now have full control of where the plot axes are placed, and we can easily add more than one axis to the figure:

```
fig = plt.figure()
graph1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])  # main axes
graph2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])  # inset axes
# main figure
graph1.plot(x, y, 'r' )
graph1.set_xlabel( 'x' )
graph1.set_ylabel( 'y ' )
graph1.set_title( 'Title\n' )
# insert
graph2.plot(y, x, 'g' )
graph2.set_xlabel( 'y' )
graph2.set_ylabel( 'x' )
graph2.set_title( 'Inset Title' );
# To save a figure to a file, we can use the savefig method in the Figure class:
fig.savefig( "filename.png" )
```

seaborn — statistical data visualization

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. Homepage for the Seaborn project: http://stanford.edu/~mwaskom/software/seaborn/

bokeh — web-based interactive visualization

Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, but also deliver this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications. Homepage for Bokeh: http://bokeh.pydata.org/

NumPy Arrays

NumPy arrays (numpy.ndarray) are the fundamental data type in NumPy. They have:

- shape
- an element type called dtype For example:

```
M, N = 5, 8
arr = np.zeros(shape=(M,N), dtype=float)
arr
arrzeros = np.zeros(30)
print(arrzeros.dtype, arrzeros.shape)
arrzeros

arrzeros2 * np.zeros((30,2))
print(arrzeros2.dtype, arrzeros2.shape)
arrzeros2
```

is the NumPy array corresponding to the two-dimensional matrix: (Broken image) NumPy has both a general N-dimensional array and a specific N-dimensional matrix data type. NumPy arrays may have an arbitrary number of dimensions. NumPy arrays support vectorized mathematical operation,

```
arr = np.arange(15).reshape(3,5)
print(          "original: "                    )
print(arr)

print()

print(          "elementwise computed:"                          )
print(((arr+4)*2) % 30)
print(arr.dtype)
((arr.reshape(15,)+4)*2) % 30
((arr.reshape(5,3)+4)*2) % 30
arr.reshape(4, 7)
```

Array Shape

Many array creation functions take a shape parameter. For a 1D array, the shape can be an integer.

```
print(np.zeros(shape=5, dtype=np.floatl6))
```

For nD arrays, the shape needs to be given as a tuple.

```
print(np.zeros(shape=(4,3,2), dtype=float), end=vsep)
print(repr(np.zeros(shape=(4,3,2), dtype=float)))
print(np.zeros(shape=(2,2,2,2)))
```

Array Types

All arrays have a specific type for their associated elements. Every element in the array shares that type. The NumPy terminology for this type is dtype. The basic types are bool, int, uint, float, and complex. The types may be modified by a number indicating their size in bits. Python's built-in types can be used as a corresponding dtype. Note, the generic NumPy types end with an underscore ("_") to differentiate the name from the Python built-in.

Python Type	NumPy dtype
bool	np.bool_
int	np.int_
float	np.float_
complex	np.complex_

Here is one example of specifying a dtype :

```
arr = np.zeros(shape=(5,), dtype=np.float_)                    # NumPy default sized fioat
print(arr, arr.dtype)
```

Watch out, though!

```
array np.array([4192984799048971232, 3, 4], dtype=np.int16)
array
np.float16(          'nan'       )
```

Check the default type…

```
np.array([1], dtype=int).dtype
```

Now we'll take just a moment to define one quick helper function to show us these details in a pretty format.

```
def    dump_array          (arr):
    print(       "%s array of %s:"                % (arr.shape, arr.dtype))
    print(arr)
    vsep =         "\n----------\n"
```

Array Creation

NumPy provides a number of ways to create an array.

np.zeros and np.ones

```
zrr = np.zeros(shape=(2,3))
dump_array(zrr)

print(np.ones(shape=(2,5)))
one_arr = np.ones(shape=(2,2), dtype=int)
dump_array(one_arr)
```

np.empty

np.empty is lightning quick because it simply requests some amount of memory from the operating system and then does nothing with it. Thus, the array returned by np.empty is uninitialized. Consider yourself warned, np.empty is very useful if you know you are going to fill up all the (used) elements of your array later.

```
# DANGER! uninitiaiized array
# (re-run this ceii and you will very likely see different values)
err np.empty(shape^(2,3), dtype=int)
dump_array(err)
```

np.arange

np.arange generates sequences of numbers like Python's range built-in. Non-integer step values may lead to unexpected results; for these cases, you may prefer np.linspace and see below. (For a quick — and mostly practical — discussion of the perils of floating-point approximations, see https://docs.python.org/2/tutorial/floatingpoint.html).

a single value is a stopping point

two values are a starting point and a stopping point

three values are a start, a stop, and a step size

As with range, the ending point is not included.

```
print( "int arg: %s" % np.arange(10), end=vsep) # cf.range(stop)
print( "float arg: %s" % np.arange(10.0), end=vsep) # cf.range(stop)
print( "step: %s" % np.arange(0, 12, 2), end=vsep) # end point excluded
print( "neg. step: %s" % np.arange(10, 0, -1.0))
```

np.linspace

np.linspace(BEGIN, END, NUMPT) generates exactly NUMPT number of points, evenly spaced, on [BEGIN, END] [BEGIN,END]. Unlike Python's range and np.arange, this function is inclusive at

BEGIN and END (it produces a closed interval).

```
print(      "End-points are included:"                  , end=vsep)
print(np.linspace(0, 10, 2), end-vsep)
print(np.linspace(0, 10, 3), end=vsep)
print(np.linspace(0, 10, 4), end=vsep)
print(np.linspace(0, 10, 20), end=vsep)
```

Diagonal arrays: np.eye and np.diag

np.eye(N) produces an array with shape (N,N) and ones on the diagonal (an NxN identity matrix).

```
print(np.eye(3))
```

Arrays from Random Distributions

It is common to create arrays whose elements are samples from a random distribution. For the many options, see:

help(np.random)

scipy

Uniform on [0,1)

```
print(      "Uniform on [0,1):"             )
dump_array(npr.random((2,5)))
```

Standard Normal

np.random has some redundancy. It also has some variation in calling conventions.

standard_normal takes one tuple argument

randn (which is very common to see in code) takes n arguments where n is the number of dimensions in the result

```
print(      "std. normal - N(0,1):"             )
dump_array(npr.standard_normal((2,5)))
print(vsep)
dump_array(npr.randn(2,5))                  # one tuple parameter
```

Arrays From a Python List… and a warning!

It is also possible to create NumPy arrays from Python lists and tuples. While this is a nice capability, remember that instantiating a Python list can take relatively long compared to directly using NumPy building blocks. Other containers and iterables will not, generally, give useful results.

```
dump_array(np.array([1, 2, 3]))

print()
dump_array(np.array([10.0, 20.0, 3]))
```

Dimensionality is maintained within nested lists:

```
dump_array(np.array([[1, 2, 3],
                     [4, 5, 6]]))
print()

dump_array(np.array([[1.0, 2],
                     [3, 4],
                     [5, 6]]))
```

Accessing Array Items

Indexing

Items in NumPy arrays may be accessed using a single index composed of multiple values (broken image)

```
arr = np.arange(24).reshape(4,6)                    # random.randint(11, size=(4, 6))

print(      "the array:"          )
print(arr, end=vsep)

print(      "index [3,2] :"              , arr[3,2], end=vsep)
print(      "index [3] :"                , arr[3], end=vsep)

# non-idiomatic, creates a view of arr[3] then indexes into that copy
print(      "index[3][2]:"               , arr[3][2])
```

Compare this with indexing into a nested Python list

```
aList = [list(row)                  for     row     in      arr]
print(aList)
```

```
print(aList[3][2])

try    :
        print(aList[3,2])
except        TypeError        as   .:
        print(       "Unhappy with multi-value index"                    )
        print(       "Exception message:"                , e)
```

Slicing

We can also use slicing to select entire row and columns at once: default default

Important Differences Between Python Slicing and NumPy Slicing

Python slicing returns a copy of the original data

Changing the slice won't change the original.

NumPy slicing returns. view of the original data

Changing the slice will change the original data The NumPy Indexing Page has a lot more information.

```
print(       "array:"          )
print(arr)

print(       "\naccessing a row:"                )
dump_array(arr[2,: ])

print(       "\naccessing a column:"                    )
dump_array(arr[:,2])

print(       "\na row:"          , arr[2,:],          "has shape:"          , arr[2,:].shape)
print(       "\na col:"          , arr[:,2],          "has shape:"          , arr[:,2].shape)
```

Bear in mind that numerical indexing will reduce the dimensionality of the array. Slicing from index to index+i can be used to keep that dimension if you need it.

```
print(       "lost dimension:"                , end=       ' '      )
dump_array(arr[2, 1:4])
print(       "\nkept dimension:"                , end=       ' '      )
dump_array(arr[2:3, 1:4])
```

Region Selection and Assignment

Multiple slices, as part of an index, can select a region out of an array

```
print(        "array: "            )
print(arr)
print(        "\na sub-array:"                    )
dump_array(arr[1:3, 2:4])
```

Slices are always views of the underlying array. Thus, modifying them modifies the underlying array

```
arr = np.arange(24) . reshape(4,6)
print(        "even elements (at odd indices) of first row:"                                    )
print(arr[0,::2])                    # select every other element from first row
arr[0,::2] = -1                    # update is done in-place, no copy
print(        "\nafter assinging to those:"                                )
print(arr)
```

Working with Arrays

Math is quite simple—and this is part of the reason that using NumPy arrays can significantly simplify numerical code. The generic pattern array OP scalar (or scalar OP array), applies OP (with the scalar value) across elements of array.

```
# array OP scalar applies across all elements and creates a new array
arr = np.arange(10)
print(        " arr:"          , arr)
print(        " arr + 1:"                , arr + 1)
print(        " arr * 2:"                , arr * 2 )
print(        "arr ** 2:"                , arr ** 2)
print(        "2 ** arr:"                , 2 ** arr)

# bit-wise ops (cf. np.logical_and, etc.)
print(        " arr | 1:"                , arr | 1)
print(        " arr & 1:"                , arr & 1)

# NOTE: arr += 1, etc. for in-place

# array OP array works element-by-element and creates. new array
arr1 = np.arange(5)
arr2 = 2 ** arr1                    # makes a new array

print(arr1, arr2, arr1 + arr2, end=vsep)
print(arr1, arr2, arr1 * arr2)
```

Elementwise vs. matrix multiplications

NumPy arrays and matrices are related, but slightly different types,

```
a, b = np.arange(8).reshape(2,4), np.arange(10,18).reshape(2,4)
print(    "a"    )
print(a)
print(    "b"    )
print(b, end=vsep)

print(    "Elementwise multiplication: a * b"                    )
print(a * b, end=vsep)
print(    "Dot product: np.dot(a.T, b)"                  )
print(np.dot(a.T, b), end=vsep)
print(    "Dot product as an array method: a.T.dot(b)"                        )
print(a.T.dot(b), end=vsep)

amat, bmat = np.matrix(a), np.matrix(b)
print(    "amat, bmat * np.matrix(a), np.matrix(b)"                       )
print(    'amat'     )
print(amat)
print(    'bmat'     )
print(bmat, end-vsep)

print(    "Dot product of matrices: amat.T * bmat"                       )
print(amat.T * bmat, end-vsep)
print(    "Dot product in Python 3.5+: a.T @ b"                    )
print(amat.T @ bmat)
```

Some Additional NumPy Subpackages

np.fft — Fast Fourier transforms

np. polynomial — Orthogonal polynomials, spline fitting

np.linalg — Linear algebra

cholesky, det, eig, eigvals, inv, lstsq, norm, qr, svd

np.math — C standard library math functions

np. random — Random number generation

beta, gamma, geometric, hypergeometric, lognormal, normal, poisson, uniform, weibull

many others, if you need it, NumPy probably has it.

FFT

```
PI = np.pi
t = np.linspace(0, 120, 4000)
nrr = np.random.random

signal = 12 * np.sin(3 * 2*PI*t)                    # 3 Hz
signal += 6 * np.sin(8 * 2*PI*t)                    # 8 Hz
```

```
signal += 1.5 * nrr(len(t))                                    # noise

# General FFT calculation
FFT = abs(np.fft.fft(signal))
freqs = np.fft.fftfreq(signal.size, t[l] - t[0])
plt.plot(t, signal); plt.xlim(0, 4); plt.show()
pit.plot(freqs, FFT);

# For one-dimensional real inputs Me can discard the negative frequencies
FFT = abs(np.fft.rfft(signal))
freqs = np.fft.rfftfreq(signal.size, t[1] - t[0])
pit.plot(freqs, FFT); pit.xlim(-0.2, 10);
```

Testing speedup of discarding negative frequencies

```
%%timeit
FFT = abs(np.fft fft(signal))
freqs np.fft.fftfreq(signal.size, t[1] - t[0])
%%timeit
FFT = abs(np.fft.rfft(signal))
freqs = np.fft.rfftfreq(signal.size, t[l] - t[0])
```

SciPy - Library of scientific algorithms for Python

The SciPy framework builds on top of the low-level NumPy framework for multidimensional arrays, and provides. large number of higher-level scientific algorithms. Some of the topics that SciPy covers are:

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems (scipy.sparse)
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- File 10 (scipy.io)

Each of these submodules provides a number of functions and classes that can be used to solve problems in their respective topics. To access the SciPy package in a Python program, we start by importing everything from the scipy module…or only import the subpackages we need…

Fourier transform

Fourier transforms are one of the universal tools in computational physics; they appear over and over again in different contexts. SciPy provides functions for accessing the classic FFTPACK library from NetLib, an efficient and well tested FFT library written in FORTRAN. The SciPy API has a few additional convenience functions, but overall the API is closely related to the original FORTRAN library.

To use the fftpack module in a python program, include it using:

```
import scipy.fftpack as spfft
# General FFT caiculation
FFT = abs(spfft.fft(signal))
freqs = spfft.fftfreq(signal.size, t[1] - t[0])
plt.plot(t, signal); plt.xlim(0, 4); plt.show()
plt.plot(freqs, FFT);
```

NumPy FFT vs. SciPy FFT vs. FFTW vs. MKLFFT

Which FFT library should you use? If you want to use the MKL, you must use NumPy. The default installations of NumPy and SciPy use FFTPACK FFTW is faster than FFTPACK, and often faster than MKL

Installing PyFFTW

1. Install FFTW

    apt-get install libfftw3-3 Iibfftw3-dev

    yum install fftw-devel

1. pip install pyfftw

Interpolation

Interpolation is simple and convenient in SciPy: The interpid function, when given arrays describing X and Y data, returns an object that behaves like. function that can be called for an arbitrary value of x (in the range covered by X). It returns the corresponding interpolated y value:

```
import scipy.interpolate as spinter
def f (x):
    return np.sin(x)

n = np.arange(0, 10)
x = np.linspace(0, 9, 100)
```

```
y_meas = f(n) + 0.1 * np.random.randn(len(n))                    # simulate measurement with noise
y_real = f(x)

linear_interpolation = spinter.interpld(n, y_meas)
y_interpl = linear_interpolation(x)

cubic_interpolation = spinter.interpld(n, y_meas, kind=                    'cubic'      )
y_interp2 cubic_interpolation(x)

fig, ax = pit.subplots(figsize=(10. 4))
ax.plot(n, y_meas,                'bs'     , label=        'noisy data'              )
ax.plot(x, y_real,                'k'    , lw2, label=           'true function'                )
ax.plot(x, y_interpl,                   'r'     , label=        'linear interp'                )
ax.plot(x, y_interp2,                   'g'     , label=        'cubic interp'                )
ax legend(loc=3);
```

COMP3321 (U) Python Visualization

Updated over 1 year aqo by [DELETED] in COMP 3321 (U) This notebook gives an overview of three visualization methods within Python, matplotlib, seaborn, and bokeh.

Python Visualization

This notebook will give a basic overview of some tools to visualize data. Much of this material is developed by Continuum Analytics, based on a tutorial created by Wesley Emeneker. First, install necessary packages:

```
# Run this if on LABBENCH
import       ipydeps

packages = [          'matplotlib'          ,    'seaborn'          ,    'bokeh'       ,    'pandas'          ,    'numpy'       ,    'scipy'          ,
'holoviews'          ]

ipydeps pip(packages)

import       numpy      as     np
import       pandas       as     pd
```

Visualization Choices

There are many different visualization chioces within Python. In this notebook we will look at three main options:

- Matplotlib
- Seaborn
- Bokeh

Each of these options are built with slightly different purposes in mind, so choose the option which best suits your needs!

Matplotlib

Matplotlib is an excellent 2D and 3D graphics library for generating scientific figures. Some of the many advantages of this library include:

- Easy to get started
- Support for LaTeX formatted labels and texts
- Great control of every element in. figure, including figure size and DPI.
- High-quality output in many formats, including PNG, PDF, SVG, EPS, and PGF.
- GUI for interactively exploring figures and support for headless generation of figure files.useful for batch jobs).

Matplotlib is well suited for generating figures for scientific publications because all aspects of the figure can be controlled programmatically. This important for reproducibility, and convenient when one needs to regenerate the figure with updated data or change its appearance.

More information at the Matplotlib web page.

To get started using Matplotlib in a Python program, import the matplotlib.pyplot module under the name plt:

```
import matplotlib.pyplot as plt
# This Line configures matplotlib to show figures embedded
# in the notebook, instead of opening. new window for each
# figure. More about that later. If you are using an old
# version of IPython, try using.Xpylab inline' instead.
%matplotlib inline
```

The matplotlib MATLAB-like API

A great way to get started with plotting using matplotlib is to use the MATLAB-like API provided by matplotlib. It is designed to be compatible with MATLAB's plotting functions, so if you are familiar with MATLAB, start here.

```
x = np.linspace(0, 5, 100)
y = x ** 2
print(x[0:10])
print(y[0:10])
plt.figure()
plt.plot(x, y, 'g' )
plt.xlabel( 'x' )
plt.ylabel( 'y' )
plt.title( 'title' )
plt.show()
plt.subplot(1,2,1)
plt.plot(x, y, 'r--' )
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-' );
```

The matplotlib object-oriented API

The main idea with object-oriented programming is to have objects to which one can apply functions and actions, and no object or program states should be global (such as the MATLAB-like API). The real advantage of this approach becomes apparent when more than one figure is created, or when a figure contains more than one subplot.

To use the object-oriented API, we start out very much like in the previous example, but instead of creating. new global figure instance, we store. reference to the newly created figure instance in the fig variable, and from it we create. new axis instance axes using the add_axes method in the Figure class instance fig :

```
fig = plt.figure()
graph = fig.add_axes([0, 0, 1, 0.3])                    # Left, bottom, width, height (range 0 to 1)
graph.plot(x, y, 'r' )
graph.set_xlabel( 'x' )
graph.set_ylabel( 'y' )
graph.set_title( ' title ' );
```

Although a bit more code is involved, the advantage is that we now have full control of where the plot axes are placed, and we can easily add more than one axis to the figure:

```
fig = plt.figure()
graph1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])             # main axes
graph2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])             # inset axes
# main figure
graph1.plot(x, y, 'r' )
graph1.set_xlabel( 'x' )
graph1.set_ylabel( 'y ' )
graph1.set_title( 'Title\n' )
```

```
# insert
graph2.plot(y, x,  'g' )
graph2.set_xlabel(  'y' )
graph2.set_ylabel(  'x' )
graph2.set_title(  'Inset Title' );
```

Plotting categorical data

Note: this works in matplotlib 2.0.0; in version 2.1, you can enter the categorical data directly on many of the matplotlib plotting methods.

```
# initioiize our data here, turning this into a list of names and a list of counts
data ={  'apples'  : 10,  'oranges'  : 15,  'lemons'  : 5,  'limes'  : 20}
names = list(data. keys())
values = list(data.values())
# first have to create numeric values to cover the axis with the categorical data
N = len(names)
ind = np.arange(N)
width = 0.35
# this will make three separate plots to demonstrate
fig, axs = pit.subplots(1, 3, figsize=(15, 3), sharey=  True )
axs[0].bar(ind + width, values)
axs[1].scatter(ind + width, values)
axs[2].plot(ind + width, values)
# here we'll space out the tick marks appropriateLy and repLace the numbers with the names
# for the labels
for  ax  in  axs:
    ax.set_xticks(ind + width)
    ax.set_xticklabels(names)

fig.suptitle(  "Categorical Plotting" )
pit.show(fig)
```

In Matplotlib 2.1+, we can do this directly

```
from  bokeh.sampledata.autompg  import  autompg  as  df
fig = plt.figure()
graph = fig.add_axes([0.1, 0.1, 2.0, 0.8])
num_vehicles = 8
graph.bar(
df[  'name' ].value_counts().index[:num_vehicles],
df[  'name' ].value_counts().values[:num_vehicles]
)
graph.set_title(  'Number of vehicles' )
graph.set_xlabel(  'Vehicle name' )
graph.set_ylabel(  'Count' )
```

```
plt.show()
```

To save a figure to a file, we can use the savefig method in the Figure class:

```
fig.savefig( "filename.png" )
```

The real power of Matplotlib comes with ploting of numerical data, and so we will wait to delve into Matplotlib further until we talk more about mathematics in Python.

seaborn - statical data visualization

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

The homepage for the Seaborn project on the internet is here .

```
import seaborn as sns
import pandas as pd
sns.set()

fig = plt.figure()
graphl = fig.add_axes([0.1, 0.1, 0.8, 0.8])                    # main axes
graph2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])                    # inset axes

# main figure
graphl.plot(x, y, 'r' )
graphl.set_xlabel( 'x' )
graphl.set_ylabel( 'y' )
graphl.set_title( 'Title\n' )

# insert
graph2.plot(y, x, 'g' )
graph2.set_xlabel( 'y' )
graph2.set_ylabel( 'x' )
graph2.set_title( 'Inset Title' );
```

More examples!

```
import random
df = pd.DataFrame()
df[ 'x' ] = random.sample(range(1,100),25)
df[ 'y' ] = random.sample(range(1,100),25)
df.head()
```

Scatterplot

```
sns.lmplot( 'x' , 'y' ,data=df,fit_reg= False );
```

Density Plot

```
sns.kdeplot(df.y);
```

Contour plots

```
sns.kdeplot(df.y, df.x);
```

Distribution plots

```
sns distplot(df.x);
```

Histogram

```
pit.hist(df.x,alpha=1.3)
sns.rugplot(df.x);
```

Heatmaps

```
sns.heatmap([df.y, df.x], anncrt= True , fmt= "d" )j
```

Bokeh

Bokeh is a Python interactive visualization library whose goal is to provide elegant graphics in the style of D3.js while maintaining high-performance interactivity over large or streaming datasets. Bokeh is designed to generate web-based interactive plots, and as such, it may not be able to provide as fine a resolution as Matplotlib. The homepage for the Bokeh project on the internet is here.

There are multiple options for displaying Bokeh graphics. The two most common methods are output_file() and output_notebook():

> The output_notebook() method works with show() to display the plot within a Jupyter notebook.

The output_file() method works with save( ) to generate a static HTML file. The data is saved with the plot to the HTML file.

In this notebook, we will focus on output_notebook().

```
from bokeh.plotting import figure, output_notebook, show
from bokeh.resources import INLINE
output_notebook(resources=INLINE)
import holoviews as hv
hv.extension( 'bokeh' )
```

First, we will make a Bokeh plot of a line. The line function takes a list of x and y coordinates as input.

```
# set up some data
import numpy as np
x = np linspace(0, 4*np.pi, 100)
y = np.sin(x)
#plot a line
plot = figure()
plot.line(x, y)
show(plot)
```

Styling and Appearance

The 'line' above is an example of an object called.Glyph'. Glyphs are made of 'lines' and 'filled areas'. Style arguments can be passed to any glyph as keywords. Some properties include:

Line properties: line_color, line_alpha, line_width, and line_dash .

Fill properties: fill_color and fill_alpha .

Bokeh uses CSS Color Names .

Here is another example showing styling options:

```
x = np.linspace(0, 4*np.pi, 100)
y = np.sin(x)

plot = figure(title= "Sine Function" )
plot.xaxis.axis_label= 'x'
plot.yaxis.axis_label= 'amplitude'

plot.line(x, y,
    line_color= 'blue' ,
    line_width=2,
    legend= 'sin(x)' )
```

```
plot.circle(x, 2*y,
        fill_color= 'red' ,
        line_color= 'black' ,
        fill_alpha=0.2,
        size=10,
        legend= '2sin(x)' )

#Line_dash is an aribrary length list of lengths
# alternating in [color, blank, color, …]
plot.line(x, np.sin(2*x),
        line_color= 'green' ,
        line_dash=[10,5,2,5],
        line_width=2,
        legend= 'sin(2x)' )

show(plot)
```

Charts

Bar charts

The Bar high-level chart can produce bar charts in various styles. Bar charts are configured with a DataFrame data object, and a column to group. This column will label the x-axis range. Each group is aggregated over the values column and bars are show for the totals:

```
from bokeh.sampledata.autompg import autompg as autompg
autompg.head()
hp_by_cyl = autompg.groupby( 'cyl' , as_index= False ).agg({ 'hp' : np.mean})
p = figure(title= "Average HP by CYL" , plot_width=600, plot_height=400)
p.vbar(x= 'cyl' , top= 'hp' , width=0.5, source=hp_by_cyl)
show(p)
```

Categorical Bar Chart

For a categorical bar chart, we still use p.vbar as above, but the top value will be the counts for the items in x. For the below example, we used the columnDataSource class from bokeh.models to actually store the data, which we can then pass in p.vbar under the source keyword. We also imported a color palette from bokeh.palettes to use as the color palette, passed in with the color keyword in our columnDataSource. (Note: when using color palettes, you need to make sure there are enough colors in the palette to cover all the values in your data.)

```
from bokeh.models import columnDataSource
from bokeh.palettes import Spectral6
fruits = [ 'Apples' , 'Pears' , 'Nectarines' , 'Plums' , 'Grapes' , 'Strawberries' ]
```

```
counts = [5,3,4,2,4,6]
source = columnDataSource(data=dict(fruits-fruits, counts-counts,
color=Spectral6))

p = figure(x_range=fruits, y_range=(0,max(counts)+3), plot_height=250,
title= "Fruit Counts" , toolbar_location= None , tools= "" )

p.vbar(x= 'fruits' , top= 'counts' , width=0.9, color= 'color' ,
legend= 'fruits' , source=source)

p.xgrid.grid_line_color = None
p.legend.orientation = 'horizontal'
p.legend.location = 'top_center'
show(p)
```

Histograms

Simple histogram using Holoviews

Using holoviews, we can easily create. histogram on top of Bokeh with the hv.Histogram function.

```
%%output size=150
%%opts Histogram (fill_color= '#CD5C5C' , line_color= 'black' )
hist = hv.Histogram(np.histogram(autompg[ 'mpg' ], bins=20), kdims=[ 'mpg' ], extents=(7, 0, 48,
hist
```

More complicated histogram using native Bokeh syntax

We can also use the quad method. In this example below, we're actually using the np.histogram function to create our histogram values, which are then passed into the quad method to create the histogram.

```
import scipy.special
import numpy as np
from bokeh.layouts import gridplot
p = figure(title= "Normal Distribution (mu=0, sigma=0.5)" , tools= "save" ,
bac kground_fill_color= "#E8DDCB" )
mu, sigma = 0, 0.5
measured = np.random.normal(mu, sigma, 1000)
hist, edges = np.histogram(measured, density= True , bins=50)
x = np.linspace(-2, 2, 1000)
p.quad(top=hist, bottom=0, left=edges[ :-1], right=edges[1:],
fill_color= "#036564" , line_color= "#033649" )
p.xaxis.axis_label= 'x'
p.yaxis.axis_label= 'Pr(x)'
show(p)
```

Scatter plots

```
%%output size=150
scatter = hv.Scatter(autompg.loc[:, [ 'mpg' , 'hp' ]])
scatter
```

Curves

```
%%output size=150
accel_by_hp = autompg.groupby( 'hp' , as_index= False ).agg({ 'accel' : np.mean})
%opts Curve [height=200, width=400, tools=[ 'hover' ]]
%opts Curve (color= 'red' , line_width=1.5)
curve = hv.Curve(accel_by_hp)
curve
```

Spikes

```
%%output size=150
spikes = hv.Spikes(accel_by_hp)
spikes
```

Using layouts to combine plots

As simple as using + to add plots together

```
%%output size=120
%%opts Curve[height=200, width=400, xaxis= 'bottom' ]
%%opts Curve(color= 'red' , line_width=1.5)
%%opts Spikes[height=200, width=400, yaxis= 'left' ]
%%opts Spikes(color= 'black' , line_width=0.8)
layout = curve + spikes
layout
```

A taste of advanced Bokeh features

Bokeh is loaded with wonderful features. Here are two final examples with no explanation. See

these notebooks for additional Bokeh information.

```
from bokeh.sampledata.iris import flowers
flowers.head()
from bokeh.models import BoxZoomTool,ResetTool,HoverTool
## Add a new Series mapping the species to a coior
colormap = { 'setosa' : 'red' , 'versicolor' : 'green' , 'virginica' : 'blue' }
```

```
flowers[ 'color' ] = flowers[ 'species' ].map( lambda x: colormap[x])

tools = [BoxZoomTool(),ResetTool(),HoverTool()]

plot = figure(title = "Iris Morphology" , tools=tools)
plot.xaxis.axis_label = 'Petal Length'
plot.yaxis.axis_label = 'Petal Width'
plot.circle(
    flowers[ "petal_length" ],
    flowers[ "petal_width" ],
    color=flowers[ "color" ], # assign the coior to each circie
    fill_alpha=0.2, size=10 )

show(plot)

x = np.Iinspace(0, 4*np.pi, 100)
y = np.sin(x)

plot = figure(tools= 'reset,box_select,lasso_select,help' )
plot.circle(x, y, color= 'blue' )
show(plot)
```

Module: Pandas

Updated over 1 year aqo by [DELETED] in COMP 3321 (U) This modules covers the Pandas package in Python, for working with dataframes.

Pandas Resource & Examples

(Note: this was modified from the Pandamonium notebook by [DELETED] on nbGallery.) This resource should help people who are new to Pandas and need to explore capabilities or learn the syntax. I'm going to provide. few examples for each command. introduce. It's important to mention that these are not all the commands available!

If you prefer video tutorials, here's. Safari series => Data Analysis with Python and Pandas

Also note that Pandas documentions is available in DevDocs .

First we'll import and install all necessary modules,

```
import ipydeps
modules = [ 'pandas' , 'xlrd' , 'bokeh' , 'numpy' ,
'requests' , 'requests_pki' , 'openpyxl' ]
ipydeps.pip(modules)
```

"pd" is the standard abbreviation for pandas, and "np" for numpy

```
import math
import pandas as pd
import numpy as np
#This is oniy inciuded to give us a sampLe dataframe to work with
from bokeh.sampledata.autompg import autompg as df
```

Creating a DataFrame

The very basics of creating your own DataFrame. I don't find myself creating them from scratch

often but I do create empty DataFrames like seen a few times further down in the guide.

```
#Create Empty DataFrame Object
df1 = pd.DataFrame()

#This is the very basic method, create empty DataFrame but specify 4 columns and their names
#You can also specify datatypes, index, and many other advanced things here
df1 = pd.DataFrame(columns=( 'column1' , 'column2' , 'column3' , 'column4' ))

#Create testing DataFrames (a, b, c), always useful for evaluating merge/join/concat/append operations.
a = pd.DataFrame([[1, 2, 3], [3,4,5]], columns=list( 'ABC' ))
b = pd.DataFrame([[5,2,3],[7,4,5]], columns=list( 'BDE' ))
c = pd.DataFrame([[11. 12,13],[17,14,15]], columns=list( 'XYZ' ))

a

b

c
```

Reading from and Writing To Files

Super easy in Pandas

CSV

Let's write our autompg dataframe out to csv first so we have one to work this. Note: if you leave

the index parameter set to True, you'll get an extra column called "Unnamed: 0" in your CSV.

```
df.to_csv( "autompg.csv" , index= False )
```

Now reading it in is super easy.

```
df1 = pd.read_csv( "autompg.csv" )
df1.head()
```

If the file contains special encoding (if it's not english for example) you can look up the encoding you need for your language or text and include that when you read the file.

```
df1 = pd.read_csv( 'autompg.csv' , encoding = 'utf-8-sig' )
```

You can also specify which columns you'd like to read in (if you'd prefer a subset).

```
df2 = pd.read_csv( ' autompg.csv' , usecols=[ 'name' , 'mpg' ])
df2.head()
```

If your file is not a csv, and uses alternative seperators, you can specify that when you read it in. Your file does not need to have a ".csv" extension to be read by this function, but should be a text file that represents data.

For Example, if you have a .tsv, or tab-delimited file you can specify that to pandas when reading the file in.

```
df1.to_csv( "autompg.tsv" , index= False , sep= '\t' )
df1 = pd.read_csv( 'autompg.tsv' , sep= '\t' )
df1.head()
```

Chunking on Large CSVs

Often times, when working with very large CSVs you will run into errors. There are. few methods to work around these errors outside of. Help Desk ticket for more memory.

If you don't have enough memory to directly open an entire CSV, as when they start going above 500MB-1GB+, you can sometimes alleviate the problem by chunking the in-read (opening them in smaller pieces).

Note: your numeric index will be reset each time.

first we'll create a Large DataFrame for an example

```
large_df = pd.DataFrame()
```

```
for i in range(100):
    # ignore_index prevents the index from being reset with each DataFrame added
    large_df = large_df.append(df1, ignore_index= True )
large_df.to_csv( "large_file.csv" , index= False )

#chunk becomes the temporary dataframe containing the data of that chunk size
for chunk in pd.read_csv( 'large_file.csv' , chunksize=1000):
    print(chunk.head(l))
```

Another chunking variation

If you still need to load a very large CSV into memory for deduplication or other processing reasons, there are ways to do it. This method uses a temporary DataFrame for appending, which gets dumped into a master DataFrame after 200 chunks have been processed. Clearing the temporary DataFrame every 200 chunks reduces memory overhead and improves speed during the append process.

You can improve efficiency by adjusting chunksize and the interval that it dumps data into the master DataFrame. There may be more efficient ways to do this, but this is effective. At the end of the cell, we have. DataFrame df1 which has all the data that we couldn't read all at once.

Notes: I use ignore_index in order to have unique index values, since append will automatically preserve index values.

```
df1 = pd.DataFrame()
df2 = pd.DataFrame()
for counter, chunk in enumerate(pd.read_csv( 'large_file.csv' , chunksize=1000)):
        #Every 200 chunks, append df2 to df1, cLear memory, start an empty df2
        if (counter % 200) == 0:
                df2 = df2.append(chunk, ignore_index= True )
                df1 = df1.append(df2, ignore_index= True )
                df2 = pd.DataFrame()
        else :
                df2 = df2.append(chunk, ignore_index= True )
#Anything Leftover gets appended to master dataframe (df1)
df1 = df1.append(df2, ignore_index= True )
#remove the temporary DataFrame
del df2
print( "There are {} rows in this DataFrame." .format(len(df1)))
df1.head()
```

Excel

Use Excelwriter to write. DataFrame or multiple DataFrames to an Excel workbook.

```
df2 = pd.DataFrame([{ 'Name' : 'Po' , 'Occupation' : 'Dragon Warrior' },
{ 'Name' : 'Shifu' , 'Occupation' : 'Sensei' }])
# this just initiaLizes the workbook
writer = pd.ExcelWriter( "test_workbook.xlsx" )
# write as many DataFrames as sheets as you want
df1.to_excel(writer, "Sheet1" )
df2.to_excel(writer, "Sheet2" )
writer.save() # .save() finishes the operation and saves the workbook
```

When reading from an Excel workbook, Pandas assumes you want just the first sheet of the workbook by default.

```
df1 = pd.read_excel( 'test_workbook.xlsx' )
df1.head()
```

To read. specific sheet, simply include the name of the sheet in the read command.

```
df1 = pd.read_excel( 'test_workbook.xlsx' , sheet_name= 'Sheet2' )
df1.head()
```

Loading from JSON/API

This is just a very simple example to show that it's very easy for JSON or API payloads to be converted to a DataFrame, as long as the payload has a structured format that can be interpreted. Pandas can write a DataFrame to a JSON file, and also read in from a JSON file,

```
df.to_json( "json_file.json" )
from_json = pd.read_json( "json_file.json" )
from_json.head()
```

The same can be done for JSON objects instead of files.

```
json_object = df.to_json() # don't specify a file and it will create a JSON object
from_json = pd.read_json(jsonobject)
from_json.head()
```

DataFrame Information Summaries

Now that your data is imported, we can get down to business. To retrieve basic infromation about your DataFrame, like the shape.column and row numbers), index values.row identifiers), DataFrame info (attributes of the object), and the count.number of values in the columns),

```
df.shape
df.index
df.info()
df.count()
```

Describe DataFrame

Summary Statistics - DataFrame.describe() will try to process numeric columns by running: (count, mean, standard deviation.std), min, 25%, 50%, 75%, max) output will be that summary.

```
df.describe()
```

Checking Head and Foot of DataFrame

Note: You can use this on most operations (especially in this guide) to get a small preview of the output instead of the entire DataFrame.

```
#Show first 5 rows of DataFrame
df.head()
#Specify the number of rows to preview
df.head(10)
#Show Last 5 rows of DataFrame
df.tail()
#Or Specify
df.tail(10)
```

Checking DataTypes

It's important to know how your DataFrame will treat the data contained in specific columns, and how it will read in the columns. Pandas will attempt to automatically parse numbers as int or float, and can be asked to parse dates as datetime objects. Understanding where it succeeded and where an explicit parse statement will be needed is important, the dataframe can provide this information.

Note: Pandas automatically uses numpy objects.

```
#View column names and their associated datatype
df.dtypes
#Select columns where the datatype is float64 using numpy (a decimal number)
df.select_dtypes([np.float64])
#Select columns where the datatype is a numpy object (like a string)
df.select_dtypes([np.object])
```

```
#Change the data type of a column
df2 = df.copy()
df2[ 'mpg' ] = df2[ 'mpg' ].astype(str)
df2[ 'mpg' ].unique()
```

Modifying DataFrames

Modifications only work on assignment or when using inplace=True, which instructs the DataFrame to make the change without reassignment. See examples below.

Change by assignment

```
df2 = df.drop( 'cyl' , axis=1)
df2.head()
```

Change in place

```
df2.drop( 'hp' , axis=1, inplace= True ) #inplace
df2.head()
```

View and Rename columns

Check all column names or Rename specific columns

```
#Check column Names
df.columns
#Store column names as a list
x = list(df.columns)
```

Batch renaming columns requires a dictionary of the old values mapped to the new ones.

```
df2 = df.rename(columns={ 'mpg' : 'miles_per_gallon' ,
                          'cyl' : 'cylinders' })
df2.head()
```

Create New columns

Similar to a dictionary, if a column doesn't exist, this will automatically create it

```
#Will populate entire column with value specified
df2 = df.copy()
df2[ 'year' ] = '2617'
```

```
df2.head()
```

Accessing Index and columns

Access a specific column by name or row by index Change the column placeholders below to actually see working columns Available in Practice DataFrame: (mpg, cyl, displ, hp, weight, accel, yr, origin, name)

```
#By column
df[ 'name' ].head()
#Alternotively and equivaient to above, this won't work if there are spaces in the column name
df.name.head()
#By Numeric index, below is specifying 2nd and 3rd rows of values
df.iloc[2:4]
#By Index + column
df.loc[[1], [ 'name' ]]
```

Remove Duplicates

Important operation for reducing a DataFrame!

Change the column placeholders below to actually see working

columns Available in Practice DataFrame: (mpg, cyl, displ, hp, weight, accel, yr, origin, name)

```
len(df)
#first Let's create some dupiicates
df2 = df.append(df, ignore_index= True )
print( "There are {} rows in the DataFrame." .format(len(df2) ))
#Remove any rows which contain dupiicates of another row
df2.drop_duplicates(inplace= True )
print( "There are now {} rows in the DataFrame." .format(len(df2)))
#or specify coiumns to reduce the number of cells in a row that must match to be dropped
df2 = df2.drop_duplicates(subset=[ 'mpg' ])
print( "There are now {} rows in the DataFrame." .format(len(df2)))
```

Filtering on columns

Filter a DataFrame based on specific column & value parameters. In the example below, we are creating a new DataFrame (df2) from our filter specifications against the sample DataFrame (df).

```
#Created new dataframe where 'cyl' value == 6
df2 = df.loc[df[ 'cyl' ] == 6]
df2.head()
# use reset_index to re-number the index values
```

```
df2 = df.loc[df[ 'cyl' ] == 6].reset_index(drop= True )
df2.head()
# not that we don't need a loc for these operations
df2 = df[df[ 'mpg' ] >= 16].reset_index(drop= True )
df2.head()
```

Fill or Drop the NaN or null values

Repair Empty values or 'NaN' across DataFrame or columns Change the column placeholders below to actually see working columns Available in Practice DataFrame: (mpg, cyl, displ, hp, weight, accel, yr, origin, name) Note: df.dropna & df.fillna are modifications and will modify the sample DataFrame. Remove "inplace=True" from entries to prevent modification

```
df.reindex?
#first we'll add some empty values
df3 = pd.DataFrame([{'name': 'Ford Taurus'}, {'mpg': 18.0}])
df2 = df.append(df3, ignore_index=True)
#Check for NaN values
df2.loc[df2['name'].isnull()]
#True/False Output on if columns contain null values
df2.isnull().any()
#Sum of all missing values by column
df2.isnull().sum()
#Sum of all missing values across all columns
df2.isnull().sum().sum()
#Locate all missing values
df2.loc[df2.isnull().T.any()]
#Fill NaN values
df2.fillna(0).tail()
#Drop NaN values
df2.dropna().tail()
#Alternatively target a column
df2['cyl'].fillna(0).tail()
#Drop row only if all columns are NaN
df2.dropna(how='all').tail()
#Drop if a specific number of columns are NaN
df2.dropna(thresh=2).tail()
#Drop if specific columns are NaN
df2.dropna(subset=['disp1', 'hp']).tail()
```

Simple Operations

```
#All Unique values in column
df[ 'mpg' ].unique()
#Count of Unique values in column
df[ 'cyl' ].value_counts()
#Count of all entries in column
```

```
df[ 'hp' ].count()
#sum of all column values
df[ 'hp' ].sum()
#mean of all column values
df[ 'cyl' ].mean()
#median of all column values
df[ 'cyl' ].median()
#min (lowest numeric value) of all column values
df[ 'cyl' ].min()
#max (highest numeric value) of all column values
df[ 'cyl' ].max()
#Standard Deviation of all column values
df[ 'cyl' ].std()
```

Sorting columns

Note: These are just the very basic sort operations. There are many other advanced methods (multi-column sort, index sort, etc) that include multiple arguments.

```
#Sort dataframe by column values
df.sort_values( 'mpg' , ascending= False ).head()
#Multi-column Sort
df.sort_values([ 'mpg' , 'displ' ]).head()
```

Merging DataFrames

While many of these are similar, there are specifics and numerous arguments that can be used in conjunction that truly customize the type of DataFrame joining/merging/appending/concating you're trying to accomplish. Note: I've provided more sample DataFrames.a, b, c) to help illustrate the various methods. Join/Merge act similar to SQL joins. This Wikipedia entry might help but it can take some time to learn and get comfortable with using them all.

```
#example df's
a = pd.DataFrame([[1,2,3], [3,4,5] ], columns=list( 'ABC' ))
b = pd.DataFrame([[5,2,3],[7,4,5]], columns=list( 'BDE' ))
c = pd.DataFrame([[11,12,13],[17,14,15]], columns=list( 'XYZ' ))
print(a)
print(b)
print(c)
```

Append DataFrames

Merges 2+ DataFrames, Does not care if dissimilar or similar. Can also use. list of DataFrames.

```
ab = a.append(b)
ab
```

Concatenate DataFrames

Simlar to append, but handles large lists of dataframes well.

```
abc = pd.concat([a,b,c])
abc
```

Join DataFrames

SQL-ish join operations (Inner/Outer etc), can specify join on index, similar columns may require specification

```
joined_df = a.join(b,how='left',lsuffix="_a",rsuffix="_b")
joined_df
```

Merge DataFrames

Merges 2+ DataFrames with overlapping columns. Very similar to join.

```
merged_df = a.merge(b, left_on='B', right_on='D')
merged_df
```

Iterate DataFrame

Iterating is only good for small dataframes, larger dataframes generally require apply/map and functions for efficiency You will inevitably use these methods at one point or another.

Iter Rows

Access to values is done by index rows[0] = Index rows[1] = values as pandas series (similar to a dict) rows[1][0] = First column value of row, can specify column rows[1]['column']

```
counter = 0
for row in df. iterrowsQ:
counter += 1
if counter > 15:
break
print(row[l].keys()[0])
print(row[l]['name'])
```

```
    print(row[0], row[1][0])
```

IterTuples

Faster and more effecient, access to values is slightly different from iterrows (Index is not nested).

rowtuples[0] = Index rowtuples[1] = First column value rowtuples[2] = Second column value

```
counter = 0
for rowtuples in df.itertuples():
    counter += 1
    if counter > 15:
        break
    print(rowtuples[1], rowtuples[2], rowtuples[3])
```

Pivoting on DataFrame

Create Excel style pivot tables based on specified criteria

```
#Basic Pivot
df.pivot_table(index=['mpg', 'name']).head()
#Specify for a more complex pivot table
df.pivot_table(values=['weight'], index=['cyl', 'name'], aggfunc=np.mean).head()
```

Boolean Indexing

Filter DataFrame on Multiple columns and values using Boolean index Note: The '&' in this example represents 'and' which might cause confusion. The explanation for this can also be a bit confusing, at least it caught me off guard the first few times. The '&' will create a boolean array (of True/False) which is used by the filtering operation to construct the output. When all 3 statements below return true for a row, pandas knows that we want that row in our output. The 'and' comparator functions differently than and will throw. 'the truth value for the array is ambiguous' exception.

```
df.loc[(df['cyl'] < 6) &
    (df['mpg'] > 35)].head()
# the same thing can be done with .query, for a more SQL-esque way to do it
# just beware that you can run into issues with string formatting when using this method
df.query("cyl < 6 & mpg > 35").head()
```

Crosstab Viewing

Contingency table (also known as a cross tabulation or crosstab) is a type of table in a matrix

format that displays the (multivariate) frequency distribution of the variables

```
pd.crosstab(df[ 'cyl' ],df[ 'yr' ],margins= True )
```

Example using multiple options

Note: This is an example using a combination of techniques seen above. I've also introduced a new method .nlargest

```
#Top Number of column1 Unique values based on the Mean of Numcolumn Unique values using .nlargest
df.cyl.value_counts().nlargest(math.ceil(df.mpg.value_counts().mean())).head
```

Create a new column with simple logic

Useful technique for simple operations

```
#Using.astype(str) I can treat the fLoat64 df['mpg'] column as a string and merge it with other strings
df2 = df.copy()
df2[ 'mpg_str' ] = df2[ 'name' ] + ' Has MPG ' + df2[ 'mpg' ].astype(str)
df2.head()
```

Functions on DataFrames

The fastest and most effecient method of running calculations against an entire dataframe. This will become your new method of 'iterating' over the data and doing analytics. axis = 0 means function will be applied to each column axis = 1 means function will be applied to each row Note: This is a step into more advanced techniques. Map/Apply/Applymap are the most efficient Pandas method of iterating and running functions across a DataFrame.

Map

Map applys a function to each element in. series, very like iterating,

```
def concon (x):
    return 'Adding this String to all values: ' +str(x)
df[ 'name' ].map(concon).head()
```

Apply

Apply runs a function against the axis specified. We are creating hp_and_mpg based on results of adding We are creating a New_column based on the results of summing column1 + column2

```
df2[ 'hp_and_mpg' ] =df2[[ 'hp' , 'mpg' ]] .apply(sum,axis=1)
df2.loc[:, [ 'hp' , 'mpg' , 'hp_and_mpg' , 'name' ]].head()
```

ApplyMap

Runs a function against each element in a dataframe(each 'cell')

```
df applymap(concon).head()
```

More Function Examples

```
def num_missing (x) :
    return sum(x. isnull())
#Check how many missing values in each column
df.apply(num_missing, axis=0)
#Check how many missing values in each row
df.apply(num_missing, axis=1).head()
```

Python 3 and Map

Note: Similar to zip, map can return an object (instead of a value) depending on how it's configured. For both zip and map, you can use list() to get the values.

```
def Example1 ( stuff):
    return stuff + 'THINGS'
#Try this without list, obverse the Newcolumn values which are returned as objects
df2 = df.copy()
df2[ 'Newcolumn' ] = map(Example1, df2[ 'name' ])
df2.head()
#Wow try with a list, problem solved when using this syntax
df2 = df.copy()
df2[ 'Newcolumn' ] = list(map(Example1, df2[ 'name' ]))
df2.head()
```

Advanced Multi-column Functions

Note: This is a technique to modify or create multiple columns based on a function that outputs multiple values in a tuple. I've written this to work directly with the sample DataFrame imported at the beginning of this resource guide.

Example2 outputs a tuple of (x, y, z) which we unpack from map using * and then zip inline.

```
def Example2(one, two, three):
    x = str(one)+' Text '+str(two)+' Text *+str(three)
    y = sum([one, two, three])
    z = 'Poptarts'
    return x, y, z

df2 = df.copy()

df2['Strcolumn' ], df2[ 'Sumcolumn' ], df2[ 'Popcolumn' ] = zip(*map(Example2, df2['mpg'], df2['cyl'], df2['hp

df2.head()
```

Conditionally Updating values

Use .loc to update values where a certain condition has been met. This is analogous to SET …

WHERE … syntax in SQL.

```
df2 = df.copy()
df2[ 'efficiency' ] = ""
# in SQL, "UPDATE <tablename> SET efficiency = 'poor' WHERE mpg < 10"
df2.loc[(df2.mpg < 10), 'efficiency' ] = "poor"
df2.loc[(df2.mpg >= 10) & (df2.mpg < 30), 'efficiency' ] = "medium"
df2.loc[(df2.mpg >= 30), 'efficiency' ] = "high"
df2.tail()
```

GroupBy and Aggregate

Pandas makes it pretty simply to group your dataframe on a value or values, and then aggregate

the other results. It's a little less flexible than SQL in some ways, but still pretty powerful. There's a

lot you can do in Pandas with GroupBy objects, so definitely check the documentation.

```
# setting as_index to False will keep the grouped values as
# regular columns values rather than indices
grouped_df = df.groupby(by=[ 'cyl' ])
# use .agg to aggregate the values and run specified functions
# note that we can't create new columns here
aggregated = grouped_df.agg({
    'mpg' : np.mean,
    'displ' : np.mean,
    'hp' : np.mean,
    'yr' : np.max,
    'accel' : 'mean'
})
aggregated.head()
```

COMP3321 - A bit about geos

Created almost 3 years ago by [DELETED] in COMP 3321 (U) This notebook gives an overview of some basic geolocation functionality within Python.

(U) A bit about geos

(U) This notebook touches some of the random Python geolocation functionality.

```
# Run this if on LABBENCH
# NOTE: geopandas REQUIRES running 'apk add geos gdal-dev'
# from a terminal window.
import ipydeps
packages = [ 'geopy' , 'geopandas' , 'bokeh' ]
for i in packages:
    ipydeps.pip(i)
```

(U) Measuring Distance

(U) Geopy can calculate geodesic distance between two points using the Vincentv distance or great-circle distance formulas, with a default of Vincenty available as the class geopy.distance.distance, and the computed distance available as attributes (e.g., miles, meters, etc.).

```
from geopy.distance import vincenty
fort_meade_md = (39.10211545,-76.7460704220387)
aurora_co = (39.729432,-104.8319196)
print(vincenty(fort_meade_md, aurora_co).miles, "Miles" )
```

```
from geopy.distance import great_circle
harrogate_uk = (53.9921491,-1.5391039)
aurora_co = (39.729432,-104.8319196)
print(great_circle(harrogate_uk, aurora_co).kilometers, "Kilometers" )
```

(U) Getting crazy with Bokeh and Maps!

(U) This information comes from this great notebook. (U) We can add map tiles to Bokeh plots to better show geolocation information! We will use some generic lat/lon data, found here

```
import pandas as pd
us_cities = pd.DataFrame().from_csv( 'us_cities.csv' ,index_col= None )
us_cities.head()
```

(U) Define the WMTS Tile Source

(U//FOUO) Adding the tile source is as easy as defining the WMTS Tile Source, and adding the tile to the the map. Note: you need your Intelink VPN spun up to connect to this server.

```
from bokeh.models import WMTSTileSource, TMSTileSource
```

[DELETED] (U) You also need to convert the lat and Ion to plot correctly on the mercator projection map:

```
import math
###METHODS FOR LAT/LONG TICK FORMATTING
def projDegToRad (deg):
    return (deg / 180.0 * math.pi)

def lat_lon_convert (n, lat_or_lon, isDeg=True):
    sm_a = 6378137.0
    sm_b = 6356752.314
    n = float(n)
    lon0 = 0.0
    if isDeg:
        n = projDegToRad(n)
    if lat_or_lon == 'latitude' :
        return sm_a * math.log((math.sin(n)+1.0)/math.cos(n))
        elif lat_or_lon == 'longitude' :
    return sm_a*(n-lon0)

us_cities[ 'merc_x' ] = list(map( lambda x: lat_lon_convert(x, 'longitude' ),us_cities.lng))
us_cities[ 'merely' ] = list(map( lambda x: lat_lon_convert(x, 'latitude' ),us_cities.lat))
```

(U) Finally, plot the data!

```
from bokeh.plotting import output_notebook, show
from bokeh.charts import Scatter
from bokeh.resources import INLINE
output_notebook(resources=INLINE)
p = Scatter(us_cities, x= 'merc_x' , y= 'merc_y' , title= "Positions of US Cities" )
p.add_tile(NGA_MAP) #vpn is necessary
show(p)
```

(U) GeoPandas

(U) GeoPandas is a project to add support for geographic data to pandas objects. It currently implements GeoSeries and GeoDataFrame types which are subclasses of pandas.Series and pandas.DataFrame respectively. GeoPandas objects can act on shapely geometry objects and perform geometric operations. (U) GeoPandas geometry operations are cartesian. The coordinate reference system (crs) can be stored as an attribute on an object, and is automatically set when loading from a file. Objects may be transformed to new coordinate systems with the to_crs() method. There is currently enforcement of like coordinates for operations, but that may change in the future.

```
from geopandas import GeoDataFrame
from matplotlib import pyplot as pit
from shapely.geometry.polygon import Polygon
from descartes import PolygonPatch
poly = Polygon([(1,1),(1,2),(1.5,3),(7,7),(5,4),(2,3)])
BLUE = '#6699CC'
fig = plt.figure()
ax = fig.gca()
ax.add_patch(PolygonPatch(poly, fc=BLUE, alpha=0.5, zorder=2))
ax.axis( 'scaled' )
plt.show()
```

Module: My First Web Application

Updated 11 months ago by [DELETED] in COMP 3321 (U) Module: My First Web Application

(U) A Word On Decorators

(U) We've learned that functions can accept functions as parameters, and functions can return functions. Python has a bit of special notation, called decorators, that handles the situation where you want to add extra functionality to many different functions. It's more likely that you will need to use and understand decorators than it is that you would need to write one, but you should still understand the basics of what's going on. (U) Suppose there are a several functions, all returning strings, that you want to "sign," i.e. append your name to.

```
def doubler (to_print):
    return to_print*2

def tripler (to_print):
    return to_print*3

doubler( "Hello!\n" )
```

(U) We an define a function that accepts this function and wraps it up with the functionality that we want:

```
def signer(f):
    def wrapper(to_print):
        return f(to_print) + '\n--Mark'
    return wrapper
```

(U) To reiterate: the argument to signer is a function, and the return value of signer is also a function. It is a function that takes the same arguments as the argument f passed into signer, and inside wrapper, f is called on those arguments. That is why something like this works:

```
signed_doubler = signer(doubler)
signed_tripler = signer(tripler)
signed_doubler("Hello!\n")
signed_tripler("Hello!\n")
```

(U) If we are willing to replace the original function entirely, we can use the decorator syntax:

```
@signer
def quadrupler(to_print):
return to_print*4

quadrupled("Hello!\n")
```

(U) Things get more complicated from there; in particular

- A function can have attributes. Therefore, a decorator can instrument a function by attaching local variables and doing something to them.
- A decorator that takes arguments must generate and return a valid decorator-style function using those arguments.
- A decorator may wrap functions of unknown signature by using (*args, **kwargs).

(U) All of this is useful when working with a complicated, multi-layer system, where much of the work would appear to be repetitive boiler plate. It's best to make the "business logic" (i.e. whatever makes this program unique) as clean and concise as possible by separating it from the scaffolding.

(U) The Flask Framework

(U) Flask is a "micro-framework", which means that it handles mostly just the web serving-receiving and parsing HTTP requests, and sending back properly formatted responses. In contrast, a macro-framework, e.g. Django, includes its own ORM for database operations, has an integrated framework for users and authentication, and an easy-to-configure administrative backend. Because

it offers so much, it takes a long time to get started with Django.

```
(VENV)[DELETED]$ pip install flask
(VENV)[DELETED]$ python
```

```
import ipydeps
ipydeps.pip( 'flask' )

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello ():
    return "Hello World"

app.run(host= '0.0.0.0' ,port=8999) # open ports:8000-9000
```

(U) Press <Ctrl-c> to stop your app.

(U) View Functions

(U) A view function is anything for which a route has been determined, using the @app.route decorator. It can return a variety of types-we've already seen a string, but it can also return a rendered template or a flask.Response, which we might use if we want to set custom headers.

```
from flask import request, make_response, redirect, url_for

fruit = { 'apple' : 'red' , 'banana' : 'yellow' , 'cranberry' : 'crimson' , 'date' : 'brown' }

@app.route('/fruits/')
def fruit_list () :
    fruit_str = "<br />" .join([ "A {} is {}" .format(*i) for . in fruit.itemsQ])
    form_str = """<br>Add something:
                <form method="post">
                <input type="text" name="fruit_name"x/input>
                <input type="text" name="fruit_color"x/input>
                """
    header_str = """<html><head><title>TEST</title></head><body> """
    footer_str = """</body></html>"""
    return header str + fruit str + form str + footer str

@app.route('/fruits/<name>/')
def single_fruit (name):
    if name in fruit.keys():
        return "A {} is {}" .format(name,fruit[name])
```

```
    else :
        return make_response( "ERROR: FRUIT NOT FOUND" , 404)

@app.route('/fruits/',methods=['POST',])
def add_fruit ():
    print(request.form)
    print(request.data)
    fruit[request.form[ 'fruit_name' ]] = request.form[ 'fruit_color' ]
    return redirect(url_for( 'fruit_list' ))

app.run(host= '0.0.0.0' ,port=8999)
```

(U) Templates

(U) Flask view functions should probably return HTML, JSON, or some other structured data format. As a general rule, it's a bad idea to build these responses as strings, which is what we've done in the simple example. Flask provides the Jinja2 template engine, which allows you to store the core of the responses in separate files and render content dynamically when the view is called. Another nice feature of Jinja2 templates is inheritance, which can help you create and maintain a consistent look and feel across a Flask website. (U) For a simple Flask app, templates should be located in a directory called templates alongside the application module. When operating interactively, the templates folder must be defined explicitly:

```
import os

templates = os.path.join(os.getcwd(), 'templates' )

app = Flask(__name__, template_folder=templates)
```

(U) A Jinja2 template can have variables, filters, blocks, macros, and expressions, but cannot usually evaluate arbitrary code, so it isn't. full-fledged programming language. It is expected that only small parts of the template will be rendered with dynamic content, so there is. custom syntax optimized for this mode of creation. Variables are surrounded with double curly braces: {{ '{{ variable }}' }}, and are typically injected as keyword arguments when the render_template function is called. Attributes of objects and dictionaries can be accessed in the normal way. Blocks, conditionals, for loops, and other expressions are enclosed in. curly brace and percent sign, e.g {{ '{X if condition %}…{% else %}…{% endif %}' }} .

```
from flask import render_template

@app.route( '/fruits/' )
def fruit_list ():
    return render_template( 'fruit_list.html' ,fruits=fruit)
```

```
@app.route('/fruits/<name>/')
def single_fruit (name):
    if name in fruit.keys():
        return render_template( 'single_fruit.html' ,name=name,color=fruit[name])
    else :
        return make_response( "ERROR: FRUIT NOT FOUND" , 404)

app.run(host= '0.0.0.0' ,port=8999)
```

(U) In this example, we also see how template inheritance can be used to isolate common elements and boilerplate. The templates used are

base.html

```
< html >
    < head >
        < title > Fruit Stand </ title >
    </ head >
    < body >
        {% block body %}
        {% endblock %>
    </ body >
</ html >
```

fruit list.html

```
{% extends.base.html" %}
{% block body %}
< hl > Fruit Stand </ hl >
< p > Available fruit: </ p >
< ul >
    {% for fruit in fruits.items() %}
    < li >< a href = M.fruits/{{ fruit [ 0 ] }> /">{{ fruit[0] }} </ a > ({{ fruit[l] }}) </ li >
    {% endfor %}
</ ul >
< p > Add. new fruit: </ p >
< form method = "post" >
    < label for = "fruit_name" > Name </ label > < input type = "text" name = "fruit_name" ></ input >< br >
    < label for = "fruit_color" > Color </ label > < input type = M text " name = "f ruit_color" x / inputxbr >
    < input type = ,, submit " value = "submit" />
</ form >
{% endblock %}
```

single_fruit.html

```
{% extends.base.html" %}
{% block body %}
```

```
< p > A {{ name }} is {{ color }}.                    </   p >
{% endblock %}
```

(U) Moving To Production

(U) In the real world, app.run() probably won't get the job done, because

- It isn't designed for performance,
- It doesn't handle HTTPS or PKI gracefully,
- It doesn't handle more than one request at a time.

(U) However, Flask makes our app conform to the WSGI standard, so it interoperates very easily with Python web-server containers, including uWSGI and gunicorn.

(U) A high performance stack for. production web application is:

- nginx: a fast, lightweight front-end server proxy that can receive HTTPS requests and pass them to gunicorn, taking care to add PKI authentication headers.
- supervisord: Process manager to make sure your app never dies.
- gunicorn: serves your flask app on a (closed, internal) port
- flask: framework in which you write an app
- your app: takes care of all the business logic
- database: SQLite if you don't need much, MySQL or Postgres if necessary.

(U) Another option is to use Apache with mod_wsgi instead of nginx, supervisord, and gunicorn.

Module: Network Communication Over HTTP(S) and Sockets

Updated over 2 years ago by [DELETED] in COMP 3321

(U) Module: Network Communication Over HTTP(S) and Sockets

(U) HTTP with requests

(U) There are complicated ways of interacting with the network using built-in libraries, such as urllib, urllib2, and httplib. We'll forgo those in favor of the requests library. This is included with Anaconda, but generally not with with other python interpreters. So for this notebook, you'll want to execute it on an Anaconda jupyter-notebook, not in labbench. In general, you can pip install it on other python implementations.

```
$ pip install requests
```

```
import requests
# One of the few things not yet requiring a certificate for Secure The Net.
resp = requests.get([DELETED])
print(resp. status_code)
print(len(resp.content))
print(len(resp.text))

# bytes vs. Unicode
resp.content == resp.text

resp.content
resp.url
resp.ok
resp.headers
```

(U) Other HTTP methods, including put, delete, and head are also supported by requests

(U) Setting up PKI

(U) Convert PI2 certificate to PEM

(U//FOUO) The requests module needs your digital signature certificate to be in PEM format. This section assumes you're starting with. P12 formatted certificate, which is what you commonly start with. If you can't find your P12 cert, you may be able to export it from your browser. If you use CSPid, other instructions may apply. We'll also do this in python below, so you don't have to do this now.

1. (U) Windows Start > type 'cygwin' > run Cygwin Terminal
2. (U//FOUO) Run cd /cygdrive/u/private/Certificates (or whatever directory holds your .p12)
3. (U)Run openssl pkcs12 -clcerts -in <your DS cert>.p12 -out <your DS cert>.pem

    (U) Enter your existing certificate password

    (U) Enter a new pass phrase. It's generally a good idea to re-use the .p12 password.

    (U) Confirm the new pass phrase.

(U//FQUO) Get the CA trust chain

(U//FOUO) To interact with sites over HTTPS, Python will need to know which certificate authorities to trust. To tell it that, you will need the following file.

1. (U//FOUO) Visit the PKI certificate authorities page (or "go pki" > click on "CA Chains" under

"Server Administrators")

2. (U//FOUO) Scroll down to "Apache Certification Authority Bundles" at the bottom and click to expand "All Trusted Partners Apache Bundles"
3. (U//FOUO) Right click on "AIITrustedPartners.crt" and save it into the directory holding your .p12 certificate

(U) HTTPS and PKI with requests

(U) To use PKI, you need the proper Certificate Authority and PEM-encoded PKI keys. We'll use a requests.Session object so that we only have to load these once..

Challenge: find a better algorithym than DES that dump_privatekey accepts

```
from OpenSSL import crypto
p12 = crypto. load_pkcs12(open( "sid_DS.p12" , "rb" ) .read(), b"Your PKI password" )
certfile = open( "sid_DS.pem" , "wb" )
certfile.write(crypto.dump_privatekey(crypto.FILETYPE_PEM, p12.get_privatekey(), 'DES' , b"mypkipasswor
certfile.write(crypto.dump_certificate(crypto.FILETYPE_PEM, p12.get_certificate()))
certfile.close()
```

```
import requests
ses = requests.Session()
ses.verify = 'Apache_Bundle_AllTrustedPartners.crt'
# Will take the certificate, or a tuple of the certificate and password or at Least it used to
# but the current version seems to not want to take a password string
# this avoids us getting prompted for the password
ses.cert = 'sid_DS.pem' #, b"mypkipassword",
resp = ses.get( 'https://home.web.nsa.ic.gov/' )
```

At this point you need to click over to the terminal running your notebook and respond to the

```
Enter PEM pass phrase:
```

prompt. You should only get one prompt per Session().

```
resp.headers
resp = ses.get( 'https://nbgallery.nsa.ic.gov/' )
resp.headers
```

(U) It's also easy to POST data to a web service with requests:

```
resp = ses.get([DELETED])
index1 = resp.text.find('method="post"')
index2 = resp.text.find( *</form>',indexl)
```

```
print(resp.text[index1-64:index2+7])

payload = {"Name":"[DELETED]",
                        "sid":"[DELETED]",
                        "Organization":"[DELETED]",
                        "others":"[DELETED]",
                        "message":"Just playing with Python(Jeannie said it was OK)",
                        "sendto":"[DELETED]",
                        "subject":"(U//FOUO) Testing", "redirect":"",
                        "classification" : "UNCLASSIFIED //~~FOR OFFICIAL USE ONLY~~ "}

resp = ses.post("https://siteworks.web.nsa.ic.gov/main/emailForm/", data=payload)

print(resp.text[resp.text.find( "Your form") :])
```

(U) In this exmaple, ses.cert could also be a list or tuple containing (certfile, keyfile), and keyfile can be a password-less PEM file or a PEM file and password string tuple, so you aren't prompted for your password every time.

(U) Low-level socket connections with socket

(U) Communication over a socket requires a server (which listens) and a client (which connects) to the server, so we'll need to open up two interactive interpreters. Both the server and the client can send and receive data. The server must

1. Bind to an IP address and port,
2. Announce that it is accepting connections,
3. Listen for connections.
4. Accept a connection.
5. Communicate on the established connection.

We'll run the server (immediately below) in the notebook and the client (below) in a separate python window on the system where we're running our jupyter-notebook.

```
#THIS IS THE SERVER
import          socket
sock_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)                                 # IPv4, TCP
HOST =      '127.0.0.1'
PORT = 50505            # USE YOUR OWN!

sock_server.bind((HOST, PORT))
sock_server.listen(1)
sock_conn.meta = sock_server.accept()
sock_conn.send(                 b"Hello, welcome to the server"                         )
sock_conn.recv(4096)
```

(U) The client must

1. Connect to an existing (IP address, port) tuple where a server is listening.
2. Communicate on the established connection.

So for our purposes, we'll run the following in. separate python window

```
#THIS IS THE CLIENT
import socket
sock_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
HOST, PORT = '127.0.0.1' , 50505 # must match a known server
sock_client.connect((HOST, PORT))
sock_client.recv(512)
sock_client.send( b"Thank you. I am the client" )
```

(U) Buffering, etc. are taken care of for you, mostly.

(U) Topics for Future Consideration:

- SOAP with SOAPpy and/or SUDS
- Using modules from the Standard Library
- XML-RPC
- Parsing HTML with BeautifulSoup

HTTPS and PKI Concepts

Updated over 3 years ago by [DELETED] (U) Overview of HTTPS and PKI concepts.

HTTPS and PKI Concepts

PKI is confusing, especially given the mix of internal and external uses, but there are some core concepts.

Public Key Infrastructure(PKI)

Each PKI certificate has two parts, the private key and the public key. The public key is simply an encrypted form of the private key. It is important to keep the private key secret at all costs. A compromized private key would allow someone else to pretend to be the original owner.

Establishing Trust

When you go to amazon.com, your browser receives their server certificate. But how do you know

you can trust it?

Certificate Authorities (CA's)

Buried in your browser is a long list of known certificate authorities, such as Verisign. The amazon.com server certificate has been digitally signed by one of these CA's. We know it's coming from amazon.com because only amazon can generate the corresponding public key, and only the corresponding private key can decrypt traffic sent to the public key. In other words, because your computer knows the public key is signed by a known CA, and your computer is sending data to that public key, only amazon can decrypt it because they have the corresponding private key.

PKI in the IC

The IC, including NSA, has its own certificate authorities (CA's). Furthermore, both the users and the servers have certificates (generally only servers have certificates on the outside). These certificates are signed by the IC CA's, which are visible at https://pki.web.nsa.ic.gov/pages/certificateAuthorities.shtml

Digital Signature (DS) Certificate

90% of the time, you're using your digital signature certificate. This certificate verifies that you are you to the various services you access on NSAnet. You also use your DS certificate for Secure Shell (SSH) to access systems like MACHINESHOP, LABBENCH, and OpenShift.

Key Encryption (KE) Certificate

On the rare occasion that you encrypt an e-mail, you use your KE certificate. Your browser doesn't actually need this certificate.

Key Formats

PKCS12

NSA keys come in PKCS12 (.p12) format. It contains both the public and private key. With Python, you need the OpenSSL package to use PKCS12 certificates.

PEM

PEM format is by far the most widely supported format on the outside. Many languages and framworks only support PEM, not PKCS12. However, you can convert your key from PKCS12 to PEM format using the openssl command.

To further complicate matters, many languages and frameworks only support unencrypted PEM certificates. You can unencrypt your PEM or PKCS12 certificate with the openssl command, but this is generally a no-no since it would allow anyone to masquerade as you.

PPK

PPK format is only used by PuTTY, the SSH tool for Windows. You can convert your key from PKCS12 to PPK format with the P12_to_PPK Converter tool.

PKI with Python

pypki2

Examples at https://gitlab.coi.nsa.ic.gov/pvthon/pypki2/blob/master/README.md

By Hand with ssl Package

SSL is the Secure Sockets Layer, which implements HTTPS (Hyper Text Transfer Protocol Secure)

Python 2.7.9+

```
from getpass import getpass
from urllib2 import build_opener, HTTPCookieProcessor, HTTPError, HTTPSHandler, Request
import ssl
pemPasswd = getpasswd( 'Enter your PKI password: ' )
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.load_cert_chain(pemCertFile, keyfile=pemKeyFile. password=pemPasswd)
context.load_verify_locations(cafile=pemCAFile)
opener = build_opener(HTTPCookieProcessor(), HTTPSHandler(context=context))
req = Request( 'https://wikipedia.nsa.ic.gov/en/Colossally_abundant_number' )
resp = opener.open(req)
print(resp.read())
```

Python 3.4+

```
from getpass import getpass
from urllib.request import build_opener, HTTPCookieProcessor, HTTPSHandler, Request
import ssl
pemPasswd = getpasswd( 'Enter your PKI password: ' )
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.load_cert_chain(pemCertFile, keyfile=pemKeyFile, password=pemPasswd)
context.load_verify_locations(cafile=pemCAFile)
opener = build_opener(HTTPCookieProcessor(), HTTPSHandler(context=context))
req = Request( 'https://wikipedia.nsa.ic.gov/en/Colossally_abundant_number' )
resp = opener.open(req)
```

```
print(str(resp.read(), encoding= 'utf-8' )) # read() returns bytes type, which has to be converted to str type
```

External Packages

OpenSSL

Handles PKCS12 and many other key formats, but not part of the standard library. It is included with Anaconda/Jupyter.

Requests

Supports only unencrypted PEM format. Takes care of. lot of little things for you like HTTP redirects. More on HTTP Status Codes at https://wikipedia.nsa.ic.gov/en/List of HTTP status codes

Python, HTTPS, and LABBENCH

Updated 3 months ago by [DELETED] in COMP 3321 (U//FOUO) This notebook demonstrates how to interact with web resources over HTTPS when using LABBENCH. It uses the requests_pki and rest_api modules.

##(U) HTTP with requests_pki (U) There are complicated ways of interacting with the network using built-in libraries, such as urllib, urllib2 and httplib. For basic (unsecured) interaction, we can use requests. However, with Secure The Net, almost everything is now PKI-enabled. (U) Luckily, there is a module for that! LABBENCH has native support for requests_pki, which makes it an ideal library for us.

```
pip3 install requests
```

```
import ipydeps
modules = [ 'requests-pki' , 'pypac' ]
ipydeps.pip(modules)
```

(U) Example 1: Obligatory example of requests

```
import requests
# One of the few things not yet requiring a certificate for Secure The Net.
resp = requests.get( 'http://airs.s2.org.nsa.ic.gov/' )
print( "Response code: {}" .format(resp.status_code))
print( "Length of content: {}" .format(len(resp. content)))
print( "Length of text: {}" ,format(len(resp.text)))
```

(U) That's interesting. Are content and text the same?

```
resp.content == resp.text
```

(U) It turns out that content stores the bytes of the response and text stores the Unicode of the response. Let's look at the text:

```
print(resp.text)
```

(U) That's great if we want the raw HTML…which in many cases we may. However, we can render the HTML response natively within Jupyter!

```
from IPython.display import display, HTML
display(HTML(resp.text))
```

(U) Notice that we didn't get any of the images that go along with this webpage, but for our purposes now this is sufficient.

(U) requests_pki

(U//FOUO) LABBENCH has made interacting with secure webpages trivial! That's because the requests_pki module works seamlessly with LABBENCH to pass your PKI with your request. Let's see how easy it is!

(U) Example 2: nbGallery

```
import requests_pki
sess = requests_pki.Session()
resp = sess.get( 'https://nbgallery.nsa.ic.gov/' )
resp.headers
display(HTML(resp.text))
```

(U) So maybe Jupyter isn't meant to be a full-fledged web-browser after all…

(U) Example 3: Notebook Gallery search

(U) Search the Notebook Gallery for a term, get the results back as JSON, and parse the JSON. This adds a new headers argument to the GET request. (U//FOUO) Normally a web server will respond with some default type of output. That may be application/html, application/xml, or something else. If you don't like that, you can try persuading the server to give you something else using an

Accept header. That will tell the server your preferred response format (i.e. the format you prefer to accept). Servers often support multiple formats, but not all of them,

```
import json
search_term = 'beautifulsoup'
url = "https://nbgallery.nsa.ic.gov/notebooks"
params = { 'q' : search_term, 'sort' : 'score' }
headers = { 'Accept' : 'application/json' }
resp = sess.get(url, params=params, headers=headers)
resp.url
print(resp.text)
# json.loads() will parse a JSON string into Lists and hashes
resp_parsed = json.loads(resp.text)
type(resp_parsed)
# take a look at it and find what you want
resp_parsed
# print the titLes of all notebooks that matched your search term
[ record[ 'title' ] for record in resp_parsed]
```

(U) Example 4: Using a proxy

(U) Sometimes you need a proxy set up, particularly when working with second party sites.

requests_pki and pypac make this setup quite easy!

```
import pypac
proxy = 'http://www.web.nsa.ic.gov/proxy/ipsec.pac'
uri = '[DELETED]'
sess = requests_pki.Session(pac=proxy)
params = { 'type' : 'Community' , 'activity_area' : 'AH' , 'project' : 'AH' , 'service' : 'All' }
respFromCSE = sess.get(url, params=params)
display(HTML(respFromCSE.text))
```

(U) Example 5: Post with JSON

Sometimes you'll need to 'post' data rather than do a 'get' request. The 'post' works similar to the 'get', but you'll need to specify parameters for the post and usually need to set the headers as well. This one posts the parameters as a JSON object; another common content type is application/x-www-form-url-encoded, in which you'll need to use the urllib library to URL encode your parameters prior to posting them.

```
base_url = 'https://namingstuff-mestern.apps.oso4.platform.cloud.nsa.ic.gov/'
# with this post, we're telling the host that we are sending json, and want to receive json
# the post parameters are sent in the 'data' key, and must be json in this case
status_code = 0
```

```
tries = 0
while not status_code == 200:
    resp = sess.post(
        base_url + 'GetRecord/languages/languages',
        headers={'Accept': 'application/json', 'Content-Type': 'application/json'},
        data=json.dumps({'language': {'$ne': 'English'}})
    )
    status_code = resp.status_code
    tries += 1
    if tries > 3:
        break

print(resp.status_code)
languages = json.loads(resp.text)
print(len(languages))
print(languages[0])
```

(U) rest_api

The rest_api library is another resource for accessing HTTPS pages on NSANet. Like requests_pki, rest_api takes care of all the PKI authentication for you, but this library is built to enable you to create what's called an 'API wrapper', which means that we're wrapping our own class around the API, which is designed to just make it easier to query the API and interpret the results. API, by the way, stands for Application Programming Interface, and is basically a clearly defined set of methods for communication with a given service, or rules for interacting with data housed in a web service.

In general if you want to hit a single web page, requests_pki is generally preferred because there's less overhead (you don't have to create a whole class to do it). But if you want to hit multiple pages at a website or API, then rest_api is probably the better way to go.

(U) Example 6: rest_api with TESTFLIGHT

This example shows a simple class that inherits from rest_api.AbstractRestAPl, and allows us to hit a couple of pages(called'endpoints') of the TESTFLIGHT API. Notice we set host and headers as class variables. With these set, we don't have to define them every time we make. query to a TESTFLIGHT page. For each page we just add the actual page or endpoint and the class fills in the rest of the URL.

```
ipydeps.pip('rest-api')
import rest_api
class Testflight(rest_api.AbstractRestAPI):
    host = 'https://tf-www.testflight.proj.nsa.ic.gov'
    headers = {'Accept': 'application/json'}
    def sources(self):
```

```
        "Returns a list of all sources that feed Testflight"
        endpoint =          '/SolanoService/rest/report/sources'
        return      self._get(endpoint).json()

    def search      (self, **kwargs):
        "Returns report summaries that match the given keyword arguments"
        endpoint =          '/SolanoService/rest/report/search/'
        return      self._post(endpoint, data=kwargs).json()

from    pprint      import      pprint
tf = Testflight()
pprint(tf.sources()[:3])

pprint(tf.search(originator=                    'NSA'   , fields=       "subject serial nipf"               , start=0, rows=3, sort=                'Newest'
```

(U) Other resources

(U//FOUO) Other notebooks on the Notebook Gallery that use requests (can you modify example 4 above to find them?)

(U//FOUO) pypki2, an open source module for working with your P12 certificate that originated at NSA. It's not part of Anaconda and works best in Jupyter on LABBENCH. It works with urllib, requests instead.

(U) One more comment. Be careful when you try to display the HTML from webpages…some webpages may affect things more than you want…

```
resp = sess.get(            'https://home.web.nsa.ic.gov/'                  )
display(HTML(resp.text))
```

UNCLASSIFIED //FOR OFFICIAL USE ONLY

Module HTML Processing With BeautifulSoup

Updated 9 months ago by [DELETED] (U) BeautifulSoup module for COMP3321.

(U) BeautifulSoup is a Python module designed to help you easily locate and pull information out of an HTML document (or string).

(U) A good deal of the time, maybe even the majority of the time, when you have to get your data from the interwebs you will query a web service that returns complete, well formatted responses (JSON, XML, etc). However, sometimes you just have to deal with the fact that the data you want can only be obtained by parsing a messy, probably automatically generated, web page.

(U) There are several approaches to dealing with web page parsing, and several Python packages

that can help you. In this lesson we cover one of the most common, BeautifulSoup.

(U//FOUO) If you are running this via Jupyter on Anaconda, you can import BeautifulSoup and use the requests module to do the [DELETED] example. If you want to perform the [DELETED] homepage example on Anaconda, you will need to export your signature PKT to PEM format (instructions here) and use a module that supports HTTPS such as urllib.request.

(U//FOUO) If you are running Jupyter on LABBENCH, execute the below cell to install bs4 and rest_api/

```
import ipydeps
modules = [ 'bs4' , 'rest_api' ]
ipydeps.pip(modules)
import rest_api
```

(U) Run this cell regardless of LABBENCH of Anaconda

```
import requests
from bs4 import BeautifulSoup
from IPython.display import HTML, display
```

(C) Let's grab the home page and save the table on the page as nice, parseable (is that a word?) text. Notice that we can do a simple .get() from the requests module. This is because the [DELETED] homepage is one of the very few plain HTTP sites left on the high side.

```
# We will try to connect and catch any exceptions in case things go awry
try :
    resp = requests.get( '[DELETED]' )
except : ..
    print( "Well, that didn't work!" )

#uncomment these lines if you want to see some of the helpful attributes qf the response object

# print(resp.status_code)
# print(Len(resp.content))
# print(Len(resp.text))

#If we got this for then we have a response (we are going to assume the response isn't
# "Access Denied") we take the response text ad create a BeautifulSoup object so we can
# tiptoe through our data
bsObj = BeautifulSoup(resp.text, "html.parser" )

# Also could have used bsObj.find_all, this returns a list of all the <table>'s in the HTML
tables = bsObj.findAll( "table" )
#open our output fiLe for writing
```

```
outfile = open( '[DELETED]_table.txt' , 'w' )
#Loop through our list of tabies from the findAll("table") above and go through the table
# one row (<tr>) and cell (<td>) at a time, outputting the information to the screen as csv
# and to the output file in pipe('|') delimited formats.
for table in tables:
    i = 0
    for tr in table.findAll( 'tr' ):
        i += 1
        j = 0
        for td in tr.findAll( 'td' ):
            print( '{},' .format(td.text), end= ' ' )
            outfile.write( "element{}:{}|" . format(j, td.text))
            j += 1
    outfile.write( '\n' )
    print()
outfile.close()
```

(U) Notice how we can display a hyperlink to our output - this might be handy if you don't want to go to Jupyter Home to display the file.

```
display(HTML( '<a href="{}" target="_blank">display file</a>' .format( "[DELETED]_table.txt" )))
```

(U) Now lets try something a little trickier. Let's pull down the home page and redisplay the "Current Activities" bulleted list/inline in our notebook.

```
#We are going to use the rest_api moduie here. This is a NSA specific package and has
# HTTPS support baked in. It makes pulling webpages using your PKIs a snap, even thohgh
# the package was really designed to access RESTfuL webservices and not web pages.
urlString = [DELETED]
parameters = "
headers = { 'text/html' ,
            'application/xhtml+xml' ,
            'application/xml;q=0.9' ,
            '*/*;q=0.8'
}
querystring = ""
#Create an api object for our host server
api = rest_api.AbstractRestAPI(host=urlString)
#Get the homepage from the server. If you wanted sub-pages off the server you would put
# that path in the querystring as something Like a /foider/page.html.
try :
    resp = api._get(querystring)
except :
    print( "Well that didn't work!" )
```

```
#Create our BeautifulSoup object
bsObj = BeautifulSoup(resp.text, "html.parser" )

#Use the .find method to get the <body> of the HTML document. We will drill down to our
# List from there. .find() only returns the first matching HTML tag, which is OK in this
# case because you *should* only have one <body> tag in the document
body = bsObj.find( 'body' )
```

(U//FOUO) Now for the sticky bit.

From the the Chrome brower Tools -> Developer Tools console (could have done this in Firefox as well from Tools->Web Developer->Toggle Tools) I ascertained that the path through the HTML to the bulleted list I care about is

```
section2 > div .item-container.item-container2.item-container-rss.item147067 > div .item-content
```

more succinctly, as xpath it is

```
// *[@id= "section2" ] /div [2] /div [1] /div/div/
```

but BeautifulSoup does not accept xpath (whomp, whomp). If you like to use xpath the Ixml module does. decent job of parsing HTML and does accept xpath syntax.

```
#Now I progress through the body object using the find_next method to get to the bulleted list
activities = body.find_next( 'div' , { 'id' : 'section2' }).find_next( 'div' ,{ 'class' : 'feedDisplay'
```

(U) Now we have the right element in the activities object. We can use the str() method to get the raw HTML from the object and either print it inline in the notebook or we can just print the text using the .text attribute.

```
# print(activities)
display(HTML(activities._str_ ()))
```

An easier way: using 'select'

In the 'Inspector' view in your Developer Tools, you can right-click on your desired tag and choose 'Copy Unique Selector' to copy the CSS selector path for your tag. Then you can use soup.select or soup.select_one to navigate directly to that tag, rather than crawling through the entire hierarchy to get to it. (Note: I ran this in Firefox, not sure what the right-click menu is like in Chrome)

```
selector =  ".rssEntries > li:nth-child(1) > div:nth-child(3)"
# at least for our version of bs4, you have to replace
# nth-child with nth-of-type
selector = selector.replace( "nth-child" , "nth-of-type" )
# bsObj.select would find all tags with that path
bsObj.select_one(selector)
```

Module: Operations with Compression and Archives

Updated about 2 years ago by [DELETED] in COMP 3321 (U) Module: Operations with Compression and Archives

```
user_string =  '''
name,username,city,state,zip_code,primary_workstation
'''

json_string =  '''
[{"author": "Jane Austen", "title": "Pride and Prejudice"}, {"author": "Fyodor Dostoevsky", "title": "Crime an
'''

with  open ( 'user_file.csv' , 'w' ) as f:
    f.write(user_string)

with  open( 'user_file.json' , 'w' ) as f:
    f.write(json_string)
```

zipfile

```
import  zipfile

with  zipfile.ZipFile( 'user_file.zip' , mode= 'w' ) as zf:
    zf.write( 'user_file.csv' )

zf = zipfile.ZipFile( 'user_file.zip' ) # with a filename

zf2 = zipfile.ZipFile(open( 'user_file.zip' )) # with a file or fiie-iike object
zf2 == zf

zf.namelist()

zf2.namelist()

z = zf.filelist[0]
```

```
z
```

```
z.filename, z.file_size
```

```
[(z.filename, z.file_size) for z in zf.filelist]
```

```
zf.getinfo( 'user_file.csv' )
user_file_csv = zf.open( 'userjFile.csv' , 'r' ) # returns a fiie-like object!
```

```
from csv import DictReader
user_data = [_ for . in DictReader(user_file_csv)]
print(len(user_data))
user_data[0]
```

```
user_file_csv.read()
user_file_csv.close()
zf.extract(zf.filelist[0], 'zfextract' )
```

gzip

```
import gzip
with gzip.open( 'user_file.csv.gz' , 'wt' ) as gf:
    gf.write( 'This string will be stored as text' )
```

```
gzip_users = gzip.open( 'user_file.csv.gz' ) # takes a file name, returns a fiie-like object!
x = gzip_users.readlines()
gzip_users.close()
x[:3]
```

```
gzip_users = gzip.open( 'user_file.csv.gz' , 'rt' )
g_user_dicts = list(DictReader(gzip_users))
g_user_dicts[:2]
```

```
with open( 'user_file.csv.gz' , 'rb' ) as f:
    still_gzipped = f.read()
still_gzipped[:100]
```

```
from io import StringIO
unpacked_users = gzip.GzipFile(fileobj=io.StringIO(still_gzipped)) # what if you have bytes or a file-like obj
unpacked_users.readlines()[:3]
```

tarfile

```
import tarfile
with tarfile.open( 'userfile.tar' , mode= 'w' ) as tf:
    tf.add( 'user_file.csv' )
```

```
        tf.add(         'user_file.json'                )

tarfile.is_tarfile(                 'userfile.tar'              ), tarfile.is_tarfile(                  'user_file.csv'             )

tf = tarfile.open(                  'userfile.tar'              )   # don't need to unzip first!

tf.getmembers()

tf.getnames()

u = tf.extractfile(                 'user_file.csv'             )
u2 = tf.extractfile(tf.getmembers()[1])

u.readline()

u2.read()[:150]

tf.extractall(              'from_tarball'              )
```

Module: Regular Expressions

Updated 11 months ago by [DELETED] in COMP 3321 (U) Module: Regular Expressions

(U) Regular Expressions (Regex)

(U) Now You've Got Two Problems…

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems. Jamie Zawinski, 1997 (U) A regular expression is a tool for finding and capturing patterns in text strings. It is very powerful and can be very complicated; the second problem referred to in the quote is a commentary on how regular expressions are essentially a separate programming language. As. rule of thumb, use the in operator or string methods like find or startswith if they are suitable for the task. When things get more complicated, use regular expressions, but try to use them sparingly, like a seasoning. At times it may be tempting to write one giant, powerful, super regular expression, but that is probably not the best thing to do.

(U) The power of regular expressions is found in the special characters. Some, like ^ and $, are roughly equivalent to string methods startswith and endswith, while others are more flexibile, especially . and *, which allow flexible matching.

(U) Getting Stuff Done without Regex

"mike" in "so many mikes!" "mike". startswith("mi") "mike".endswith("ke") "mike".find("k") "mike".isalpha() "mike".isdigit() "mike".replace("k", "c")

(U) Regular expressions in Python

There are only a few common methods for using the re module, but they don't always do what you would first expect. Some functionality is exposed through flags, which are actually constants (i.e. int defined for the re module), which means that they can be combined by addition.

```
import re
re.match( "c" , "abcdef" )
re.match( "a" , "abcdef" )
re.search( "c" , "abcdef" )
re.search( "C" , "abcdef" )
re.search( "C" , "abcdef" , re.I) # re.IGNORECASE
re.search( "^c" , "ab\ncdef" )
re.search( "^c" , "ab\ncdef" ,re.M) # re.MULTILINE
re.search( "^C" , "ab\ncdef" , re.M + re.I)
```

(U) In both match and search, the regular expression precedes the string to search. The difference between the two functions is that match works only at the beginning of the string, while search examines the whole string.

(U) When repeatedly using the same regular expression, compiling it can speed up processing. After. compiled regular expression is created, find, search, and other methods can be called on it, and given only the search string as a single argument.

```
c_re = re.compile( "c" )
c_re. search( "abcde" )
```

Regex Operators

. - matches a single character A - matches beginning of a string or newline $ - matches end of string

0 or more of something

1 or more of something ? - 0 or 1 of something *?, +?, ?? - don't be greedy (see example below) {3} - match 3 of something {2,4} - match 2 to 4 of something \ - escape character [lrnLRN] - match any ONE of the letters l, r, n, L, R, N [a-m] - match any ONE of letters from a to m [a|m] - match letter a or m \w - match a letter \s - match a space \d - match a digit

```
re.search ( "\w*s$" , "Mike likes cheese\nand Mike likes bees" )
re.findall( "\(\d{3}\)\s\d{3}-\d{4}" , "Hello, I am a very bad terrorist. If you wanted to know, my phone number
re.findall( "mi.*ke" , "i am looking for mike and not all this stuff in between mike" )
re.findall( "mi.*?ke" , "i am looking for mike and not all this stuff in between mike" )
```

Capture Groups

Put what you want to pull out of the strings in parentheses ()

```
my_string = "python is the best language for doing 'pro'gramming"
result = re.findall( "'\(\w+)" , my_string)
print(result)
print(result[0])
```

Matches and Groups

(U) The return value from a successful call of match or search is a match object; an unsuccessful call returns None. First, this is suitable for use in if statements, such as if c_re.search("abcde"): …. For complicated regular expressions, the match object has all the details about the substring that was matched, as well as any captured groups, i.e. regions surrounded by parentheses in the regular expression. These are available via the group and groups methods. Group 0 is always the whole matching string, after which remaining groups (which can be nested) are ordered according to the opening parenthesis.

```
m = re.match( r"(\w+) (\w+)" , "Isaac Newton, physicist" )
m.group()
m.group(1)
m.group(2)
m.groups()
```

Other Methods

(U) Other regular expression methods work through all matches in the string, although what is returned is not always straightforward, especially when captured groups are involved. We demonstrate out some basic uses without captured groups. When doing more complicated things, please remember: be careful, read the documentation, and do experiments to test!

```
re.findall( "a.c" , "abcadcaecafc" ) # returns list of strings
re.finditer( "a.c" , "abcadcaecafc" ) # returns iterator of match objects
re.split( "a." , "abcadcaecafc" ) # returns list of strings.
```

(U) The sub method returns a modified copy of the target string. The first argument is the regular expression to match, the second argument is what to replace it with-which can be another string or a function, and the third argument is the string on which the substitutions are to be carried out. If the sub method is passed a function, the function should take a single match object as an argument and return a string. For some cases, if the substitution needs to reference captured groups from the regular expression, it can do so using the syntax \g<number>, which is the same as accessing the groups method within a function.

```
re.sub( "a.*?c" , "a--c" , "abracadabra" )
re.sub( "a(.*?)c" , "a\g<1>\g<1>c" , "abracadabra" )
```

```
def reverse_first_group (matchobj) :
    match = match(obj.group()
    rev_group = matchobj.group(1)[::-1]
    return match[:matchobj.start(1)] + rev_group + match[matchobj.end(1):]
re.sub( "a(.*?)c" ,reverse_first_group, "abracadabra" )
```

(U) In the above, we used start and end, which are methods on a match object that take a single numeric argument-the group number-and return the starting and ending indices in the string of the captured group.

(U) One final warning: if a group can be captured more than once, for instance when its definition is followed by a + or a *, then only the last occurrence of the group will be captured and stored.

Hashes

Updated. months aqo by [DELETED] in COMP 3321 (U) Computing Hashes in Python

(U) Hashes

(U) Let's start with hashes. Hashes map data of arbitrary size to data of fixed size and have a variety of uses:

securely storing passwords

verifying file integrity

efficiently determining if data is the same (U) There are many different hashing algorithms. You've probably heard of some of the more common ones, such as MD5, SHA1, and SHA256.

(U) Hashes have some useful features:

they are one-way, meaning that given. hash, there isn't a function to convert it back to the original data

they map data to a fixed output, which is useful when comparing large amounts of data (such as files) (U) So let's generate a hash.

```
from hashlib import sha256
sha256( 'abc' .encode( 'ascii' )).hexdigest()
```

(U) or

```
sha256( b'abc' ).hexdigest()
```

(U) We all know storing plaintext passwords is bad. A common technique of avoiding this is to store the hash of the password, then check if the hashes match. So we can create a short function to check if the typed password matches the stored hash:

```
def check_password (clear_password, password_hash):
    return sha256(clear_password).hexdigest() == password_hash
```

(U) Does anyone know why storing the hash of a password is bad? (U) If the password hash database was ever compromised, it would be vulnerable to a pre-computation attack (rainbow table), where an attacker pre-computes hashes of common passwords. There are tools such as scrypt to help mitigate this vulnerability. (U) How about a safer use of hashes? Suppose you need to look for duplicate files? Doing a byte-per-byte comparison of every file to every other file would be really expensive. A better approach is to compute the hash of each file, then compare the hashes.

```
import os
from hashlib import md5
def get_file_checksum (filename) :
    h = md5()
    chunk_size = 8198
    with open(filename, 'rb' ) as f:
        while True :
            chunk = f.read(chunk_size)
            if len(chunk) == 0:
                break
            h.update(chunk)
    return h.hexdigest()
```

(U) There is a small danger with this approach: collisions. Since we're mapping a lot of data to a smaller amount of data, there is the possibility that two files will map to the same hash. For SHA256, the chances that two files have the same hash are 1 in 2^256, or about 1 in 1,16e+77. So even with a lot of files, the chance of a collision is small.

(U) Notice that we don't need to read in the entire file at once. One really cool feature of hashes is they can be updated:

```
h = sha256( b'abc' )
h.update( b'def' )
h.hexdigest()
sha256( b'abcdef' ).hexdigest()
```

Module: SQL and Python

Updated almost 2 years ago by [DELETED] in COMP 3321 (U) Module: SQL and Python

(U) The Odd Couple: Programming and Databases

(U) It makes a lot of sense to keep your data in a database, and programming logic in a program. Therefore, it's worth overcoming the fundamentalimpedance mismatch between the two technologies. In the most common use cases, where the program isn't too terribly complicated and the data isn't too crazily interconnected, things usually work just fine.

(U) Python has a recommended Database API, although there are slight variations in the way this API is implemented, which is one reason to use a metalibrary like SQLAIchemy (we'll get to this later). The standard library only provides an implementation for SQLite, in the sqlite3 package. Connections to other database types require external packages, such as MySQLdb confusingly, to get this you have to pip install MySQL-python).

(bobby drop tables)[broken image link]

(U) Basics with sqlite3

To interact with a database, a program must

1. Establish a connection
2. Create a cursor
3. Execute commands

    Read the results

    Commit the changes

4. Close the cursor and/or connection

(U) Using a basic adapter, commands are executed by passing strings containing SQL commands as arguments.

```
import sqlite3
conn = sqlite3.connect( 'test.db' ) # SQLite specific: creates db if necessary
```

```
cur = conn.cursor()
cur.execute(    """create table fruit (
                  id integer primary key,
                  name text not null,
                  color text default "RED"
                  )"""
                  )

cur.execute(    " insert into fruit (name) values ("apple")"                                    )    # not there yet
conn.commit()       # to make sure it's written
cur.execute(    """ select * from fruit """                         )    # returns the cursor--no need to capture it.
cur.fetchone()
```

(U) When making changes to the database, it's best to use parameter substitution instead of string subtitution to automatically protect against unsanitized input. The sqlite3 module uses ? as its substitution placeholder, but this differs between database modules (which is a major headache when writing code that might have to to connect to more than one type of database).

```
fruit_data = [(     'banana'    ,   'yellow'    ),
                ( 'cranberry'       ,   'crimson'       ),
                ( 'date'    , 'brown'       ),
                ( 'eggplant'        ,   'purple'        ),
                ( 'fig'     , 'orange'      ),
                ( 'grape'       ,   'purple'        )]

for     i   in  fruit_data:
        cur.execute(        """insert into fruit (name, color) values (?,?)"""                                  , f)

cur.execute(        """select * from fruit"""                       )    # DANGER! DATA HASN'T BEEN WRITTEN YET!

cur.fetchone()

cur.fetchmany(3)
cur.fetchall()
```

(U) A cursor is iterable:

```
more_fruit = [(     'honeydew'      ,   'green'     ),
                (   'ice cream bean'        ,   'brown '        ),
                (   'jujube'        ,   'red'       )]

cur.executemany(        """insert into fruit (name, color) values (?, ?)"""                                  , more_fruit)
cur.execute(    """select * from fruit"""                       )

[item[1]        for     item    in  cur]    # read the name

cur.execute(    'PRAGMA table_info(fruit)'                      )
```

```
for line in cur:
    print(line)

cur.fetchall()
conn.commit()              # aiways remember to commit!
```

(U) In sqlite3, many of the methods associated with a cursor have shortcuts at the level of connection behind the scenes, the module creates a temporary cursor to perform the operations. We will not cover it because it isn't portable.

(U) Other Drivers

(U) The most common databases are MySQL and Postgres. Installing the packages to interact with them is often frustrating, because they have non-Python dependencies. Even worse, the most current version of mysql-python in PYPI is broken, so we request a different version:

```
(VENV)[DELETED]$ pip install mysql-python==1.2.3
(VENV)[DELETED]$ pip install oursql
(VENV)[DELETED]$ pip install psycopg2.

Error: pgconfig executable not found.
```

(U) With enough exceptions to make life very frustrating, they work like sqlite3.

(U) SQLAIchemy

(U) SQLAIchemy is a very powerful, very complicated package that provides abstraction layers over interaction with SQL databases. It includes all kinds of useful features like connection pooling. We'll discuss two basic use cases; in both of which we just want to use it to get data in and out of Python.

(U) Cross-Database SQL

(U) Imagine the following scenario: during development you'd like to use SQLite, even though your production database is MySQL. You don't plan to do anything fancy; you already know the SQL statements you want to execute (although there are a couple of things you always wished sqlite3 would do for you, like returning a dict instead of a tuple.

(U) Enter SQLAIchemy. It does require that you have a driver installed, e.g. MySQLdb, to actually talk to the database, but it takes care of all the ticky-tack syntax details. By default, it even commits changes automatically!

```
import ipydeps
```

```
ipydeps.pip( 'sqlalchemy' )
import sqlalchemy
engine = sqlalchemy.create_engine( 'sqlite:///test.db' ) # database protocol and URL
result = engine.execute( 'select * from fruit' )
ans = result.fetchall()
first_ans = ans[0]
type(first_ans)
first_ans[0]
first_ans.keys()
first_ans.values()
engine.execute( '''insert into fruit (name) values (?)''' ,( 'kumquat' ))
engine.execute( '''insert into fruit (name,color) values (?, ?)''' ,[( 'lime' , 'green' ),( 'mango'
result = engine.execute( 'select * from fruit' )
result.fetchall()
```

(U) Now, to move to MySQL, all you have to do is use. different URL, which follows the pattern:

```
dialect+ driver: //username:password@host:port/database
```

The SQLAIchemy documentation lists all the databases and drivers.

(U) As Object Relational Mapper

(U) The real power in SQLALchemy is in using it to store and retrieve Python objects from a database without ever writing a single line of SQL. It takes a little bit of what looks like voodoo at first. We'll skip most of the details for now, at the risk of this being a complete cargo cult activity. Open up a new file called sql_fruit.py and put the following into it:

```
from sqlalchemy import create_engine, column, Integer, String, Date
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine( 'sqlite:///test.db' )
Base = declarative_base()
```

```
Session = sessionmaker(bind=engine)
db_session = Session()

class Fruit (Base):
    _tablename_ = 'fruit'
    id = Column(Integer, primary_key= True )
    name = column(String)
    color=column(String, default= "RED" )
    def __init__ (self, name, color):
        self.name = name
        self.color = color
    def __repr__ (self):
        return "<Fruit {}: {}, {}>" .format(self.id, self.name, self.color)
```

(U) Now, in the interactive interpreter:

```
from sql_fruit import *

f__query = db_session.query(Fruit)

f_query.all()
f_query.first()
nectarine = Fruit( 'nectarine' , 'orangered' )
db_session.add(nectarine)
db_session.commit()
```

Easy Databases with sqlite3

Created over 3 years ago by [DELETED] in COMP 3321 (U) Example on using sqlite3 to group and average data instead of using dictionaries.

Easy Databases with sqlite3

The great thing about sqlite3 is that it allows you to create a simple, local database without having to install any servers or other tools. The entire database is contained in a single file. Here we're going to create a simple database that holds daily stock data. This is related to the Structured Data and Dates Exercise from COMP3321 at https://jupyter-gallery.platform.cloud.nsa.ic.gov/nb/884fbd2f/Structured-Data-and-Dates-Exercise

We'll use the same AAPL stock data from Yahoo Finance available at https://urn.nsa.ic.gov/t/Ogrli

First we import the packages for reading the CSV, parsing the dates, and working with sqlite3.

```
from csv import DictReader
from datetime import datetime
```

```
import sqlite3
```

Create the Table

Here we have a function that creates a stocks table in our database (referenced by db_conn). The columns are:

symbol: Simply holds the stock ticker symbol so we can store records for more than just AAPL.

year, month, day: We break the date out into three integer columns because it's easier to do queries against precise dates or groups/ranges of dates. If the date were kept as. string, forming the query string would be much more difficult.

week: We calculate the week for each date so the data can be grouped by week.

price: This corresponds to the Adj Close column from the CSV; the closing price for that symbol on that date. When we're done forming our table, we need to commit the changes to the database,

```
def create_database (db_conn):
    cur = db_conn.cursor()
    cur.execute( 'CREATE TABLE stocks (symbol text, year integer, month integer, day integer, week integer, pr
    db_conn.commit()
```

Import the Data

This function imports data for a symbol from input_file using the database connection object db_conn. This is very similar to the Structred Data exercise, except we do a SQL insert for each record. The question marks get associated with the values in the tuple (the second argument to cursor.execute after the INSERT command string).

After we've iterated over all records for the inserts, we need to commit them to the database with db_conn.commit().

```
def import_data (symbol, input_file, db_conn):
    with open(input_file, 'r' ) as infile:
        symbol = symbol.strip().upper()
        reader = DictReader(infile)
        cursor = db_conn.cursor()
        for record in reader:
            dt = datetime.strptime(record[ 'Date' ], '%Y-%m-%d' )
            week = dt.isocalendar()[1]
            price = float(record[ 'Adj Close' ])
            cursor.execute( "INSERT INTO stocks VALUES (?, ?, ?, ?, ?, ?)" , (symbol, dt.year, dt.month, dt.day
        db_conn.commit()
```

Making SQL do all the Work

In the Structured Data Exercise, the student had to manually group the data in a dictionary by week (really a (year,week) tuple). SQL can do this for us, and even calculate the average. We break down each line of the query as follows:

SELECT: We want the year, the week, and the average for the prices for the days on that year week, so we use SELECT to pick those columns.
FROM: We're working with the stocks table, so we say FROM stocks.
WHERE: To put conditions on a SQL query, we use the WHERE clause. Here our only condition is that we only want data associated with a certain symbol (AAPL in this case).
GROUP BY: We need to group the data by week, and since we don't want the same week in two different years to get grouped together, we have to group by the year and the week, hence - GROUP BY year, week.
ORDER BY: To display our data in descending order by date, we have to use ORDER BY. SQL even allows mixed ascending and descending subgroups, so we have to specify that we want both the year and the week in descending order.

We append each result from our query into an empty list, which gets returned by our function.

```
def weekly_avenages (symbol, db_conn):
    cur = db_conn.cursor()
    results = []
    for result in cur.execute( '''SELECT year, week, avg(price)
                                  FROM stocks
                                  WHERE symbol=?
                                  GROUP BY year, week
                                  ORDER BY year DESC, week DESC''' , (symbol,)):
        results.append(result)
    return results
```

Execute!

Here we create our database in aapl.db. Note that it will raise an exception if the table has already been created in the database. If you simply comment out the create_database() call to get around this, then be careful since import_data() will insert the data again, so you'll have the double entries in your stocks table. Restart this notebook and delete aapl.db to get a truly fresh start.

```
db_file = 'aapl.db'
db_conn = sqlite3.connect(db_file)
create_database(db_conn)
import_data( 'AAPL' , 'aapl.csv' , db_conn)
```

Now that the data is in the database, we can call our weekly_averages() query function. This will just display the list of results.

```
weekly_averages( 'AAPL' , db_conn)
```

Module: Structured Data: CSV, XML, and JSON

Updated over 1 year ago by [DELETED] in COMP 3321 (U) Read, write, and manipulate CSV, XML, JSON, and Python's custom pickle and shelve formats.

(U) Setup

(U) For this notebook, you will need the following files:

- user file.csv
- user file.xml.

(U) Right-click each to download and "Save As," then, from your Jupyter home, navigate to the folder containing this notebook and click the "Upload" button to upload each file from your local system.

(U) Introduction: It's Sad, But True

(U) Much of computing involves reading and writing structured data. Too much, probably. Often that data is contained in files--not even a database. We've already worked with opening, closing, reading from, and writing to text files. We've also frequently used string methods. At first, it might seem that that's all we need to work with CSV, XML, and other structured data formats.

(U) After all, what could go wrong with the following?

```
my_csv_file = open( 'user_file.csv' , 'r' )

csv_lines = my_csv_file.readlines()

comma_separated_records = [line.split( ',' ) for line in csv_lines]

xml_formatter = """<person>
                    <name>{}</name>
                    <address>{}</address>
                    <phone>{}</phone>
                    </person>"""

xml_records = "\n" . join([xml_formatter.format(*record) for record in comma_separated_records])
```

```
xml_records = "<people>" + xml_records + "</people>"

with open( 'file.xml' , 'w' ) as f:
    f.write(xml_records)
```

(U) In a rapidly-developed prototype with controlled input, this may not cause a problem. Given the way the real world works, though, someday this little snippet from a one-off script will become the long-forgotten key component of a huge, enterprise-wide project. Somebody will try to feed it data in just the wrong way at a crucial moment, and it will fail catastrophically.

(U) When that happens, you'll wish you had used a fully-developed library that would have had a better chance against the malformed data. Thankfully, there are several-and they actually aren't any harder to get started with.

(U) Comma Separated values (CSV)

(U) The most exciting things about the csv module are the DictReader and DictWriter classes. First, let's look at the plain vanilla options for reading and writing.

```
import csv
f = open ( 'user_file.csv' )
reader = csv.reader(f)
header = next(reader)
all_lines = [line for line in reader]
all_lines.sort()
g = open( 'user_file_sorted.csv' , 'w' )
writer = csv.writer(g)
writer.writerow(header)
writer.writerows(all_lines)
g.close()
```

(U) CSV readers and writers have other options involving dialects and separators. Note that the argument to csv.reader must be an open file (or file-like object), and the reading starts at the current cursor position.

(U) Accessing categorical data positionally is not ideal. That is why csv also provides the DictReader and DictWriter classes, which can also handle records with more or less in them than you expect. When given only a file as an argument, a DictReader uses the first line as the keys for the remaining lines; however, it is also possible to pass in fieldnames as an additional parameter.

```
f.seek(0)

d_reader = csv.DictReader(f)

records = [line for line in d_reader]
```

(U) To see the differences between reader and DictReader, look at how we might extract cities from the records in each.

```
# for the object from csv.reader
cities0 = [record[2] for record in all_lines]

# for the object from csv.DictReader
cities1 = [record['city'] for record in records]

cities0 == cities1
```

(U) In a Dictwriter, the fieldnames parameter is required and headers are not written by default. If you want one, add it with the writeheader method. If the fieldnames argument does not include all the fields for every dictionary passed into the Dictwriter, the keyword argument extrasaction must be specified,

```
g = open ('names_only.csv', 'w')

d_writer = csv.Dictwriter(g, ['name', 'primary_workstation'], extrasaction='ignore')

d_writer.writeheader()

d_writer.writerows(records)

g.close()
```

(U) Javascript Object Notation (JSON)

(U) JSON is another structured data format. In many cases it looks very similar to nested Python dicts and lists. However, there are enough notable differences from those (e.g. only single quotation marks are allowed, boolean values have a lowercase initial letter) that it's wise to use a dedicated module to parse JSON data. Still, serializing and deserializing JSON data structures is

relatively painless.

(U) For this section, our example will be a list of novels:

```
import json
novel_list = []
novel_list.append({'title': 'Pride and Prejudice', 'author': 'Jane Austen'})
novel_list.append({'title': 'Crime and Punishment', 'author': 'Fyodor Dostoevsky'})
novel_list.append({'title': 'The Unconsoled', 'author': 'Kazuo Ishiguro'})
json.dumps(novel_list) # to string
with open ('novel_list.json', 'w') as f:
    json.dump(novel_list,f) # to file
the_hobbit = '{"title": "The Hobbit", "author": "J.R.R. Tolkien"}'
novel_list.append(json.loads(the_hobbit)) # from string
with open('war_and_peace.json' as f: # <-- if this file existed
        novel_list.append(json.load(f)) # from fiie
```

(U) By default, the load and loads methods return Unicode strings. It's possible to use the json module to define custom encoders and decoders, but this is not usually required.

(U) Extensible Markup Language (XML)

(U) This lesson is supposed to be simple, but XML is complicated. We'll cover only the basics of reading data from and writing data to files in a very basic XML format using the ElementTree API, which is just the most recent of at least three approaches to dealing with XML in the Python Standard Library. We will not discuss attributes or namespaces at all, which are very common features of XML. If you need to process lots of XML quickly, it's probably best to look outside the standard library (probably at a package called Ixml).

(U) Although there are other ways to get started, an ElementTree can be created from. file by initializing with the keyword argument file:

```
from xml.etree import ElementTree
xml_file = open ('user_file.xml')
user_tree = ElementTree.ElementTree(file=xml_file)
```

(U) To do much of anything, it's best to pull the root element out of the ElementTree. Elements are iterable, so they can be expanded in list comprehensions. To see what is inside an element, the ElementTree module provides two class functions: dump (which prints to screen and returns None) and tostring. Each node has a text property, although in our example these are all empty except for leaf nodes.

```
root_elt = user_tree.getroot()
users = [u for u in root_elt]
print(ElementTree.tostring(users[0]))
u_children = [x for x in users[0]]
u_children[2].text
u_children[2].text = 'north-x5-1234'
ElementTree.dump(users[0])
```

(U) To get nested descendant elements directly, use findall, which returns a list of all matches, or find, which returns the first matched element. Note that these are the actual elements, not copies, so changes made here are visible in the whole element tree.

```
all_usernames = root__elt.findall( 'user/name/username' )
[n.text for n in all_usernames[:10]]
```

(U) To construct an XML document:

make an Element,

append other Elements to it (repeating as necessary),

wrap it all up in an ElementTree, and

use the ElementTree.write method (which takes a file name, not a file object).

```
apple = ElementTree.Element( 'apple' )
apple.attrib [ 'color' ] = 'red'
apple.set( 'variety' , 'honeycrisp' )
apple.text = "Tasty"
ElementTree.dump(apple)
```

```
fruit_basket = ElementTree.Element( 'basket' )
fruit_basket.append(apple)
fruit_basket.append(ElementTree.XML( '<orange color="orange" variety ="navel" </orange>' ))
ElementTree.dump(fruit_basket)
fruit_tree = ElementTree.ElementTree(fruit_basket)
fruit_tree.write( 'fruit_basket.xml' )
```

(U) Bonus Material: Pickles and Shelves

(U) At the expense of compatibility with other languages, Python also provides built-in serialization and data storage capabilities in the form of the pickle and shelve modules.

Pickling

```
import pickle
pickleme = {}
pickleme[ 'Title' ] = 'Python is Cool'
pickleme[ 'PageCount' ] = 543
pickleme[ 'Author' ] = '[DELETED]'
with open ( '/tmp/pickledData.pick' , 'wb' ) as p:
    p = pickle.dump(pickleme, p)
with open ( '/tmp/pickledData.pick' , 'rb' ) as p:
    p = pickle.load(p)
print(p)
```

(U) Shelving

(U) Creating a Shelve

```
import shelve
pickleme = {}
pickleme[ 'Title' ] = 'Python is Cool'
```

```
pickleme[ 'PageCount' ] = 543
pickleme[ 'Author' ] = '[DELETED]'
db = shelve.open( '/tmp/shelve.dat' )
db[ 'book1' ] = pickleme
db.sync()
pickleme[ 'Title' ] = 'Python is Cool -- The Next Phase'
pickleme[ 'PageCount' ] = 123
pickleme[ 'Author' ] = '[DELETED]'
db[ 'book2' ] = pickleme
db.sync()
db.close()
```

(U) Opening a Shelve

```
db = shelve.open( '/tmp/shelve.dat' )
z = db.keys()
a = db[ 'book1' ]
b = db[ 'book2' ]
print(a)
print(b)
print(z)
db.close()
```

(U) Modifying. Shelve

```
db = shelve.open( '/tmp/shelve.dat' )
z = db.keys()
a = db[ 'book1' ]
```

```
b = db[ 'book2' ]
print(a)
print(b)
print(z)
a[ 'PageCount' ] = 544
b[ 'PageCount' ] = 129
db[ 'book1' ] = a
db[ 'book2' ] = b
db.close()
```

Module: System Interaction

Updated over 3 years ago by [DELETED] in COMP 3321 (U) Basic operating system interaction using the os, shutil, and sys modules.

(U) Introduction

(U) Python provides several modules for interacting with your operating system and the files and directories it holds. We will talk about three: os, shutil, and sys.

(U) Be aware that while this notebook is unclassified, your output may not be (depending on the files you're displaying).

(U) os Module:

(U) This module helps you interact with the operating system, providing methods for almost anything you would want to do at a shell prompt. On POSIX systems, there are over 200 methods in the os module; we will just cover the most common ones. Be aware that the os module includes methods that are not cross-platform compatible; the documentation is helpfully annotated with Availability tags. (U) Directory discovery and transversal is pretty basic:

```
import os
os.getcwd()
os.chdir( '/tmp' ) # Unix dir--choose different dir for Windows
```

```
os.listdir()

os.getcwd()

walker = os.walk(os.curdir)

type(walker)

list(walker)
```

(U) Avoid one common confusion: os.curdir is a module constant (. on Unix-like systems), while os.getcwd() is a function. Either one can be used in the method os.walk, which returns a generator that traverses the file system tree starting at the method's argument. Each successive value from the generator is a tuple of (directory, [subdirectories], [files]). (U) A variety of methods allow you to examine, modify, and create or remove directories and files,

```
f = open ( 'new_temp_file.txt' , 'w' )

f.close()

os.stat( 'new_temp_file.txt' )

os.mkdir( 'other_dir' )

os.rename( 'new_temp_file.txt' , 'other_dir/tempfile.txt' )
```

(U) The os.path submodule provides additional functionality, including cross-platform compatible methods for constructing and deconstructing paths. Note that while it is possible to join. path completely, deconstructing a path occurs one element at a time, right to left.

```
sample__path = os.path.join( 'ford' , 'trucks' , 'f150' )

sample_path

os.path.split(sample_path)

os.path.exists(sample_path)
```

(U) Information about the current environment is also available, either via specific methods or in the os.environ object, which functions like a dictionary of environment variables. If os.environ is modified, spawned subprocesses inherit the changes.

```
os.getlogin()
```

```
os.getuid()                 # Unix
os.getgroups()              # Unix
os.environ
os.environ[ 'NEW_TEMP_VAR' ] = '123456'
os.uname()          # Unix
```

shutil Module

(U) Living on top of the os module, shutil makes high-level operations on files and collections of files somewhat easier. In particular, functions are provided which support file copying and removal, as well as cloning permissions and other metadata.

```
import shutil
shutil.copyfile(src,dest)                   # overwrites dest
shutil.copymode(src,dest)                   # permission bits
shutil.copystat(srcjdest)                   # permission bits and other metadata
shutil.copy(src,dest)               # works Like cp if dest is. directory
shutil.copy2(src,dest)                # copy then copystat
shutil.copytree(src,dest)
shutil.rmtree(path)               # must be reai directoryj not. symLink
shutil.move(src,dest)               # works with directories
```

(U) sys Module

(U) The sys module provides access to variables and functions used or maintained by the Python interpreter; it can be thought of as a way of accessing features from the layer between the underlying system and Python. Some of its constants are interesting, but not usually useful.

```
import sys
sys.maxsize
sys.byteorder
sys.version
```

(U) Other module attributes are sometimes useful, although fiddling with them can introduce problems with compatibility. For instance, sys.path is a list of where Python will look for modules when import is called. If it is modified within a script, and then modules can be loaded from a new location, but there is no inherent guarantee that location will be present on a system other than your own! On the other hand, sys.exit() can be used to shut down a script, optionally returning an error message by passing a non-zero numeric argument.

Manipulating Microsoft Office Documents with win32com

Updated almost 3 years ago by [DELETED] in COMP 3321 (U) Demonstration of using win32com to create and modify Microsoft Office documents.

(U) Manipulating Microsoft Office Documents with win32com

(U) Welcome To Automation with win32com!

(U) The win32com module connects Python to the Microsoft Component Object Model interface that enables inter-process communication and object creation within Microsoft Office applications. (U) Note: win32com only exists on Windows platforms, so this notebook will not run on LABBENCH. In order to run this notebook, install Anaconda3 on your Windows platform and use jupyter-notebook.

(U) "Hello World" for Word

(U) We need to import the library, and open Word.

```
import win32com.client

word = win32com.client.Dispatch( 'Word.Application' )
```

(U) Dispatch checks to see if Word is already open. If it is, it attaches to that instance. If you'd like to always open a new instance, use DispatchEx.

(U) By default, Word will start, but won't be visible. Set this to True if you want to see the application.

```
word.Visible = True
```

(U) Create a document and add some text, setting a font size that we like.

```
worddoc = word.Documents.Add()

worddoc.Content.Text =                    "Hello World"

worddoc.Content.Font.Size = 18
```

(U) Save the document and exit the application. Note that win32com bypasses the normal Python file object, so we need to account for the Windows directory separator. (U//FOUO) Also, ClassifyTool may nag you for a classification. In order to prevent this, in Word, select the "ClassifyTool" tab, click on "Options", and under "When Closing Document", uncheck "Always show Classification Form", and click "Save".

```
worddoc.SaveAs(          'u:\\private\\jupyter\\win32com\\hello.docx'                    )
word.Quit()
```

(U) That's it!

(U) More Elaborate Word Example

(U) There's another option for starting the application:

```
word = win32com.client.gencache.EnsureDispatch(                    'Word.Application'          )
```

(U) This can take slightly longer, but enables access to win32com constants, which are required for some methods. The alternative is to look through the win32com documentation for the value of the constants you need.

(U) Let's take a look at a possible use case. Say we have reports in a particular format that we need to regularly generate. We can create a template with the sections that will be replaced. In this case, they are ReportEvent, ReportTime, and ReportPlace. First, download the template. Then open the template and create. dictionary with the sections and the data that will be used.

```
constants = win32com.client.constants                    # save some future typing

word.Visible =          True

worddoc = word.Documents.Open(                'u:WprivateWjupyter\\win32com\\demo_template.docx'                    )

event_details = {          "ReportEvent"          :    "[DELETED]"        ,
                           "ReportTime"           :    "[DELETED]"        ,
                           "ReportPlace"          :    "[DELETED]"        }
```

(U) Now the magic happens. Lets iterate through the dictionary, replacing all of the sections with the data.

```
# Execute( FindText, MatchCase, MatchWhoLeWord, MatchWiLdcards, MatchSoundsLike, MatchAllWordForms,
# Forward, Wrap, Format, RepLaceWith, RepLace)
for tag, data in event_details.items():
    _ = word.Selection.Find.Execute( tag, False , False , False , False , False , True , constants.wdFindContinue,
```

(U) We can add a couple of paragraphs of additional info, and we're done.

```
paragraph1 = worddoc.Paragraphs.Add()
paragraph1.Range.Text = 'Additional info\n'
footer = worddoc.Paragraphs.Add()
footer.Range.Text = 'Produced by me\n'
worddoc.SaveAs( 'u:\\private\\jupyter\\win32com\\demo__out.docx' )
word.Quit()
```

(U)PowerPoint

(U) PowerPoint works very similarly. Again, download the template

```
ppt = win32com.client.Dispatch( 'PowerPoint.Application' )
presentation = ppt Presentations.Open( 'u:\\private\\jupyter\\win32com\\MyTeam_template.pptx'
```

(U) Did you notice that we didn't need to set ppt.Visible ? PowerPoint is always visible,

```
title = presentation.Slides(1)
```

(U) We know the first slide is the title slide, so we've set. variable to it. PowerPoint presentations are made up of slides, which in turn are collections of shapes. To modify. presentation, we need to know which shape is which. Let's take. look at title:

```
title
```

(U) Hmm. That's not very helpful. Let's see what methods we have:

```
dir(title)
```

(U) At this point you're probably realizing that COM objects don't act like normal Python objects.

```
help(title)
```

(U) So Python just takes anything you try to do with title and passes it on to the Windows COM library. Which means you'll need to consult Microsoft's Win32Com documentation if you have questions about something.

(U) Let's get back to working with this presentation. We still need to find out which shape is which:

```
for i, shape in enumerate(title.Shapes):
    shape.TextFrame.TextRange.Text = 'Shape #{0}'.format(i+1)
```

(U) This sets the text for each shape to its index number so we now have a number associated with each shape. You only need to do this when you're writing your script. Once you create your template, the shape numbers won't change. So the title is #1 and the subtitle #2. (U) Undo few times will remove the numbers. (U) Let's update the title slide with today's date:

```
from datetime import date
today = date.today().strftime('%Y%m%d')
title.Shapes(2).TextFrame.TextRange.Text = today
```

(U) Now let's update the status of our two focus areas. We'll skill the step of identifying the shapes we want to modify,

```
focus1 = presentation.Slides(2)
focus1.Shapes(2).TextFrame.TextRange.Text = 'All Good, Boss'
focus2 = presentation.Slides(3)
focus2.Shapes(2).TextFrame.TextRange.Text = 'Sir, We have a problem'
```

(U) Now save the presentation with today's date, and Bob's your uncle.

```
presentation.SaveAs('u:\\private\\jupyter\\win32com\\MyTeam_{0}.pptx'.format(today))
presentation.Close()
ppt.Quit()
```

(U) Visio

(U) Starting the application should look familiar:

```
visio = win32com.client.Dispatch ("Visio.Application")
documents = visio.Documents
document = documents. Add("Basic Network Diagram.vst") # Start with a built-in template
```

```
document.Title = "New Network Graph" # Add a title
pages = visio.ActiveDocument.Pages
page = pages.Item(1)
```

(U) Visio is visible by default, but can be hidden if desired. (U) So we've created a document and grabbed the page associated with it. Visio shapes are part of stencil packages, so let's add a couple.

```
NetworkStencil = visio.Documents.AddEx( "periph_m.vss" , 0, 16+64, 0)
ComputerStencil = visio.Documents.AddEx( "Computers and Monitors.vss" , 0, 16+64, 0)
```

(U) Other stencils are:

- Network Locations: netloc_m.vss
- Network Symbols: netsym_m.vss
- Detailed Network shapes: dtlnet_m.vss
- Legends: lgnd_m.vss

(U) Other stencil names can be found on the Internet. (U) Now we need the shape masters that we'll use.

```
pc = ComputerStencil.Masters.Item( "PC" )
router = NetworkStencil.Masters.Item( "Router" )
server = NetworkStencil.Masters. Item( "Server" )
connector = NetworkStencil.Masters.item( "Dynamic Connector" )
```

(U) The names match the names you see when you view the shapes in the stencil sidebar. Let's add a few shapes.

```
pc1 = page.Drop(pc, 2, 2)
pc1.Text = "10.1.1.1"
pc2 = page.Drop(pc, 10, 10)
pc2.Text = "10.1.1.2"
server1 = page.Drop(server, 15, 5)
server1.Text = "10.1.1.100"
router1 = page.Drop(router, 8, 8)
router1.Text = "10.1.1.250"
```

(U) Some of the shapes went off the page, so resize. You can wait until the end to do this, but it's more fun to watch the connections being drawn.

```
page.ResizeToFitContents()
page.CenterDrawing()
```

(U) Now draw the connectors.

```
arrow = page.Drop(connector, 0, 0)
arrowBegin = arrow.CellsU(          "BeginX"       ).GlueTo(pc1.CellsU(          "PinX"      ))
arrowEnd = arrow.CellsU(          "EndX"      ).GlueTo(router1.Cellsu(          "PinX"      ))
arrow.Text =          "pc1 connection"
```

(U) We can customize a connector

```
arrow = page.Drop(connector, 0, 0)
arrowBegin = arrow.CellsU(          "BeginX"       ).GlueTo(pc2.CellsU(          "PinX"      ))
arrowEnd = arrow.CellsU(          "EndX"      ).GlueTo(router1.Cellsu(          "PinX"      ))
arrow.CellsU(          "LineColor"          ).Formula =          "=RGB(255, 153, 3)"
arrow.CellsU(          "EndArrow"          ).Formula =          "=5"
arrow.CellsU(          "EndArrowSize"          ).Formula =          "=4"
arrow.CellsU(          "LineWeight"          ).FormulaU =          "=5.0 pt"
arrow.Text =          "pc2 connection"
arrow = page.Drop(connector, 0, 0)
arrowBegin = arrow.CellsU(          "BeginX"       ).GlueTo(serverl.CellsU(          "PinX"      ))
arrowEnd = arrow.CellsU(          "EndX"      ).GlueTo(routerl.Cellsu(          "PinX"      ))
arrow.Text =          "server1 connection"
```

(U) Now resize, recenter, and save.

```
page.ResizeToFitContents()
page.CenterDrawing()
document.SaveAs(          'U:\\private\\jupyter\\win32com\\visio_demo.vsdx'          )
```

(U) Close the application.

```
visio.Quit()
```

(U) win32com works with Excel too, but due to the slowness of the interface, you're probably better off using pandas.

Module: Threading and Subprocesses

Updated over 2 years ago by [DELETED] in COMP 3321 (U) Module: Threading and Subprocesses

(U) Module: Threading and Subprocesses

(U) Concurrence and Python's GIL - i.e. Python doesn't offer true concurrence

(U) Python's Global Interpreter Lock (GIL) means that you can really only have one true thread at one time. However, Threading in Python can be immensely helpful in speeding up processing when your script can perform subsequent steps that do not depend on the ouptut of other steps. Basically, it gives the illusion of being able to do two (or more) things at the same time.

(U) Threading

(U) Threading allows you to spawn off "mini programs" called threads that work independently of the main program (sort of). Threading allows you to send data off to a function and let it work on getting results while you go on with your business. It can also allow you to set up functions that will process items as you add them to work queue. This could be especially helpful if you have parts of your program that take a long time to execute but are independent of other parts of your program. A good example is using a thread to execute a slow RESTful web service query.

(U) This adds some complexity to your life. Threads act asynchronously - meaning that you have limited control as to when they execute and finish. This can cause problems if you are depending on return values from threads in subsequent code. You have to think about if and how you need to wait on thread output which adds extra things to worry about in terms of accessing data. Python provides a thread-safe container named Queue. Queues will allow your threads access without becoming unstable, unlike other containers (such as dictionaries and lists) which may become corrupted or have unstable behavior if you access them via multiple threads.

(U) Subprocess

(U) The subprocess module is useful for spinning off programs on the local system and letting them run independently.

```
import ipydeps
modules = [ 'threading' , 'queue' ]

for i in modules:
    installed_packages = [package.project_name for package in ipydeps._pip.get_installed_distributions()]
    if (m not in installed_packages) and (m not in ipydeps.sys.modules):
        ipydeps.pip(m)

import time
from threading import Thread, Timer, Lock
from queue import Queue
import random
```

```
result_q = Queue()
work_q = Queue()
work_list = []
# The worker thread pulls an item from the queue and processes it

def worker ():
    while True :
        item = work_q.get()
        do_work(item)
        work_q.task_done() ttpause white untit current work_q task has compteted

def do_work (item) :
    ## submit query process resutts and add resutt to Queue
    result_q.put( wait_random(item) )

def wait_random (t) :
    time.sleep(t[1])
    print( 'finished task {}' .format(t[0]))

def hello () :
    print( "hello, world" )
    # loading up our work__q and work__tist with the same random ints between 1 and 10
    time_total = 0
    for i in range(10):
        x = random.randint(1,10)
        time_total += x
        work_q.put((i,x))
        work_list.append((i,x))
        work_q.qsize()
    len(work_list)

%%time
print( 'This should take {} seconds' .format(time_total))
for w in work_list:
        wait_random(w)

%%time
for i in range(5):
        t = Thread(target=worker)
        t.daemon = True # thread dies when main thread exits . If we don't do this, then the threads will continue
        # "Listen" to the work_q and take items out of the work_q and automatically process as you
        # stick more items into the work_q
        t.start() # you have to start. thread before it begins to execute
work_q.join() # block until all tasks are done
```

(U) You can also use the Timer class to specify that a thread should only kick off after a set amount of time. This could be critical if you need to give some other treads a head start of for various other reasons. Remember, when we are doing threading you have to keep timing in mind!

```
%%time
```

```
# stupid little example
ti = Timer(5.0, hello)
ti.daemon =  True
ti.start()  # after 5 seconds, "hello, world" will be printed
```

(U) You can mix these. The output below will most likely look like a bucket of crazy because threads execute (sort of) independently.

```
#loading up our work_q and work_list with the same random ints between 1 and 10
for  i  in  range(10):
    x = random.randint(1,10)
    work_q.put((i,x))

%%time
for  i  in  range(5):
    t = Thread(target=worker, )
    t.daemon =  True
    t.start()

ti = Timer(5.0,hello)
ti.daemon =  True
ti.start()  # ti *will probably* print 'hello, world' before all the other threads finish,
# or it might not it depends on the work_q contents
work_q.join()  # block until all tasks are done
```

(U) Subprocesses

(U) For most subprocess creation you will usally want to use the subprocess.run() convenience method. Please note, if you wish to access the STDOUT or STDERR output you must specify a value for the stdout and stderr arguments. Using the subprocess.PIPE constant puts the results from STDOUT and STDERR into the CompletedProcess object's attributes.

```
import  subprocess

completed = subprocess.run([  'ls'  ,  '-l'  ], stdout=subprocess.PIPE, universal_newlines=  True  )

print(  "ARGS:"  , completed.args)
print(  "STDOUT:"  ,completed.stdout)
print(  "STDERR:"  ,completed.stderr)
print(  "return code:"  , completed.returncode)

completed = subprocess.run([  'ls'  ,  'nosuchfile'  ], stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_new

print(  "ARGS:"  , completed.args)
print(  "STDOUT:"  , completed.stdout)
print(  "STDERR: "  , completed.stderr)
print(  "return code:"  , completed.returncode)
```

```
dir(completed)

type(completed)
```

(U) For finer-graned control you can use subprocess.Popen(). This allows greater flexability, and allows you to do things like kill spawned subprocesses, but be careful - you may get some unexpected behavior.

```
completed = subprocess.Popen([ 'ls' , '-l' ], stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newline

print( "ARGS: " , completed.args)
print( "STDOUT: " , completed.stdout)
print( "STDERR: " , completed.stderr)
print( "return code: " , completed.returncode)

dir(completed)

type(completed)
```

(U) If you are just looking for a quick way to dump the output from the subprocess into a variable, you can use subprocess.check_output(). This is an older way of doing a specific type of .run() so you will see it used in Python 2. It takes many of the same parameters as .run() but has a few extra that correspond more closely to Popen()

```
completed = subprocess.check_output([ 'ls' , '-l' ],universal_newlines= True )

dir(completed)
print(completed)
type(completed)
```

Distributing a Python Package at NSA

Updated 2 months ago by [DELETED] in COMP 3321 (U//FOUO) Directions for how to make and distribute a Python package using the nsa-pip server.

UNCLASSIFIED //FOR OFFICIAL USE ONLY (U//FOUO) Module: Distributing a Python Package at NSA (U//FOUO) At NSA, internally-developed Python packages can be installed by configuring pip to point to the nsa-pip server (https://pip.proj.nsa.ic.gov/). But how do you push your own package out to nsa-pip? The basic steps are as follows:

1. (U) Make a python package.
2. (U) Make the package into a distribution.

3. (U) Push the project to GitLab.

4. (U//FOUO) Add a webhook for nsa-pip to the GitLab project.

5. (U//FOUO) Push the package to nsa-pip.

1. (U) Make a python package

(U) Recall from an earlier lesson that a python package is just a directory structure containing one or more modules, a special __init__.py file, and maybe some nested subpackages. The name of the package is just the name of the directory.

awesome/ | --__init__.py | -- awesome.py

(U) Example module

(U) Here are the contents of the awesome module (awesome.py) from before:

```
class Awesome (object) :
    def __init__ (self, awesome_thing):
        self.thing = awesome_thing
    def __str__ (self):
        return "{0.thing} is awesome!!!" .format(self)

def cool (group):
    return "Everything is cool when you're part of {0}" .format(group)

if __name__ == '__main__' :
    a = Awesome( "Everything" )
    print(a)
```

(U) __init__.py

(U) Once again, the __init__.py file needs to be present but can be empty. It is intended to hold any code needed to initialize the package. If the package has many subpackages, it may define __all__ to specify what gets imported when someone uses the from <package name> import * syntax. Other things __init__.py commonly includes are import statements for subpackages, comments/ documentation, and additional code that glues together the subpackages.

(U) Since our module is small and we have nothing to initialize, we'll leave __init__.py empty. If we wanted, we could actually put all the awesome.py code in __init__.py itself, but that can be confusing for other developers. We could also define:

```
__all__= [ 'Awesome' , 'cool' ]
```

(U) But since those would get imported anyway, we don't need to do that. You only need to define __all__ if the package is complex enough that you want to import some things and not others or ensure that subpackages get imported.

(U//FOUO) For another example, see the __init__.py file for the parserror module on GitLab.

2. (U) Make the package into a distribution

(U) At this point, you could tar or zip up that package, give it to someone else, and they could extract it and use it. But what directory should they put the package in? What if your package depends on other modules to work? What if you change the package and want to keep track of the version? These considerations suggest that there is some package management that needs to go on. Python does have the pip package manager to handle a lot of that. So how do we distribute the package so you can just pip install awesome?

(U) First we need to add another layer of stuff around our package to help set it up for installation.

awesome/ |-- README.md |-- awesome/ |-- init.py |-- awesome.py |-- setup.cfg |-- setup.py

(U) You can see that our original package directory has been moved down to a subdirectory. Above it we have a new project directory by the same name, and alongside it there are a few more files. Let's look at each.

(U) setup.py

(U) setup.py is the most important file. It mainly contains a setup() function call to configure the distribution. It is also used to run the various packaging tasks via python setup.py. The setup() function comes from the setuptools package, which is not part of the Python standard library. You may need to pip install setuptools first to use it. (setuptools improves the legacy distutils package that is part of the standard library and is the officially recommended distribution tool these days.)

(U//FOUO) Our setup.py looks like this:

```
from setuptools import setup
setup(
        version='1.0.0',
        name='awesome',
        description='(U) An awesome module for awesome things.',
        long_description=open('README.md', 'r').read()
        url='https://gitlab.coi.nsa.ic.gov/python/awesome.git',
        author='COMP3321',
        author_email= 'comp3321@nsa.ic.gov,
        scripts=[],
        packages=['awesome',],
        package_data={},
```

```
        install_requires=[]
)
```

(U) This just scratches the surface of the arguments you can give to setup(). You can find more details on the outside from the Python Packaging Authority (https://packaging.python.org/tutorials/distributing-packages/). For example, you can specify which versions of Python are compatible using the classifiers and python_requires options, specify dependencies with install_requires and other *_requires options, and include non-code data files.

(U//FOUO) For another example, see parserror's setup.py.

(U) setup.cfg

(U) setup.cfg is an INI file that configures some defaults for the setup.py options. Ours is fairly simple:

```
[metadata]
description-file = README.md
```

(U) If you are using wheel, you may also want to set this if your code works for both Python 2 and 3:

```
[bdist_wheel]
univeral =1
```

(U) README.md

(U) README.md is just a Markdown file that gives an overview of what the package is for and describes how to install and use it. (You may also see README.rst files instead. The .rst stands for restructured text, which is a popular Python-specific way to format text.)

3. (U) Push the project to GitLab

(U//FOUO) Of course, with a project like this, you should be using version control. And since the nsa-pip server ties into GitLab, you'll need to use Git. Learning Git could be a course in itself, so we'll just cover the basics here.

(U) For those who are unfamiliar, Git is a distributed version control manager. It's version control because it allows to to track and revert changes in your text files easily in a Git repository. You can view the history of your changes as a tree, and even branch off from it and merge back in easily if you need to. It's distributed because everybody who has a copy of the git repository has a full copy

of the history with all the changes. And it's a manager (like pip is a manager) because it manages all the tracking itself and gives you. bunch of commands to let you add and revert your changes.

a. (U) Install and configure Git

(U//FOUO) With any luck, your system will already have Git installed. If you're working on LABBENCH, it should already be there. If you're on another Linux system, you can yum install git (for CentOS/RHEL) or apt-get install git (for Ubuntu). If you're on Windows, you will probably want to use Git Bash or TortoiseGit or something similar. [DELETED] There are additional Windows directions here.

(U//FOUO) After Git is installed, you will need to configure it and SSH to connect to GitLab. On LABBENCH, the easiest way to do that is to run this (Ruby) notebook to set up Git and SSH in jupyter-docker. It takes care of:

- making a .gitconfig,
- making an overall .gitignore, and
- making an encrypted RSA key pair based on your PKI cert for use with SSH

(U//FOUO) If you are on another system, you will need to take care of those things yourself. For tips, see the one-time setup instructions for Git on Wikilnfo. [DELETED]

b. (U) Add version control to your project

(U) Now to work! Enter your local project directory (i.e. the top level where setup.py lives) and run git init. That will turn your project into a Git repository and get Git ready to track your files. (Note that it doesn't actually start tracking them yet-you have to explicitly tell it what to track first.)

(U) Next, since this is a Python package, you probably want to a .gitignore file to your project directory containing:

```
build/
*.pyc
```

(U) That will tell Git to ignore temporary files made when you run setuptools commands. (U) Now our project structure should look something like this:

```
awesome/
|-- .git/
| `--(…git stuff…)
|-- .gitignore
|-- README.md
|-- awesome/
        |-- __init__.py
        |-- awesome.py
```

```
|-- setup.cfg
|-- setup.py
```

(U) Next, you can start tracking all these files for changes by running:

```
git add *
git commit -m "Initial commit."
```

c. (U) Make a corresponding GitLab project

(U) Congratulations! Your package is now under version control. However, you have the only copy of it, so if your local copy goes away, your code is gone. To preserve it and enable others to work on it, you need to push it to GitLab.

(U//FOUO) The first step is to make a new GitLab project:

1. (U) Visit the new project page on GitLab
2. (U) Enter your package name as the Project path (e.g. awesome).
3. (U//FOUO) Enter the overall classification level of the code and files in your project.
4. (U) Leave the Global access level set to Reporter. That will allow others to both clone (copy) your code and file issues if there are problems with your package. See the GitLab permissions chart for a full description of the access levels.
5. (U//FOUO) Choose the Namespace. By default it will make a personal project under your name and SID. If you belong to a GitLab group (and have the right permissions), you can also add the project to that group. Consider joining the Python group and adding your project there.
6. (U) Add a short description of your package, if you want.
7. (U//FOUO) Select a visibility level. Since NSA GitLab uses PKI, there is no difference between "internal" and "public".
8. (U) Hit "Create Project."

(U) Afterwards, you can copy the URL from the main page of your new project and put it in the url argument to setup() in setup.py.

d. (U) Push your code out to GitLab

(U) Your new project page should have some instructions on how to push code from an existing folder or Git repository. Near the top of the page you should see a box with "SSH" highlighted to the left and a git@gitlab.coi.nsa.ic.gov ... address in the box to the right. Copy that address. Then,

1. (U) cd to the top level of your project
2. (U//FOUO) git remote add origin git@gitlab.coi.nsa.ic.gov … (using the .git address you just copied)

3. (U) git push -u origin master

(U) Afterwards, if you visit your GitLab project page, you should see a list of your project files and the rendered contents of your README.

4. (U//FOUO) Create a tag for the package release.

(U) Back in your local repository, tag your local project with pip- and the release version and push that tag to GitLab. For example, if your first version is 0.1.0, run:

```
git tag -a pip-0.1.0 -m "Releasing to the pip repo"
git push origin pip-0.1.0
```

(U) Ideally, version numbers should comply with Python's PEP 440 specification. In plain English, that means they should be of the form:

```
Major.Minor.Micro
^ ^ ^
| | \- changes every bugfix
| \- changes every new feature
|
\- changes every time backwards-compatibility broken
```

5. (U//FOUO) Create distributions

(U) Source distribution (sdist) and Wheel distributions are both collections of files and metadata that can be installed on a machine and then used as a Python library. Wheels are considered a "built distribution" while sdist needs one extra build step, although that is transparent to a user if you are using pip to install. For most use cases, you want to build both an sdist and bdist_wheel, upload both, and let pip work out which one to use (almost always the Wheel).

(U) An example of building and publishing the distributions is as follows:

```
pip3 install setuptools wheel twine
python3 setup.py sdist bdist_wheel
```

a. (U) Upload package to NSA-Pypi with Twine--

(U) twine is a Python library developed by the same team that maintain Pypi. It uses requests.py to make secure (https) uploads of Python packages, and offers some command-line arguments to make it easy to specify what repository, client cert, ca bundle, and username/password to use. The username and password can be blank (or anything you want) for uploading to NSA-Pypi, because

NSA-Pypi will use your PKI and Casport to determine authentication/authorization.

(U) Note that the NSA-Pypi server supports XPE with Labbench, so using twine there on Labbench does not require uploading your personal cert. In non-Labbench environments, the following command should work:

```
twine upload dist/* -u " -p " \
    --repository-url https://nsa-pypi.tools.nsa.ic.gov \
    --cert /path/to/ca_bundle.pem \
    --client-cert.path/to/your_cert.pem \
    --verbose
```

Bonus!

Python Packaging Authority

(U) For years there were a variety of ways to gather and distribute packages, depending on whether you were using Python 2, Python 3, distutils, setuptools, or some obscure fork that tried to improve on one or another of them. This eventually got so messy that some developers got together and formed the Python Packaging Authority (PyPA) to establish best practices and govern submissions to PyPI. PyPA's approach is now referenced in the official Python documentation. They can be found at http://packaging.python.org/ and have many useful tutorials and guides, including how to use setuptools and push a package to PyPI itself.

(U) Testing

(U) Consider using common test modules like doctest, unittest, and nose to add tests to your code. You can try test-driven development (TDD) or even behavior-driven development (BDD) with behave.

(U) Documentation

(U) Make sure your code includes docstrings where appropriate, as well as good comments and a helpful README.

(U) virtualenv

(U) To solve the problem of coming up with an isolated, repeatable development environment (i.e. make sure you have the dependencies you need and that they don't conflict with other Python programs), most developers use virtualenv and virtualenvwrapper.

(U) wheel

(U) If your package is large or needs to compile extensions, you may want to distribute. pre-built "wheel". The wheel module adds some functionality to setuptools if you pip install wheel. On the receiving end, when you pip install packages, you may occasionally see that pip is actually installing a .whi file -- that's a wheel. It is relatively new and is the recommended replacement for the old .egg format. UNCLASSIFIED //FOR OFFICIAL USE ONLY

Module: Machine Learning Introduction

Created 3 months ago by [DELETED] in COMP 3321 (U//FOUO) This notebook gives COMP3321 students an introduction to the world of machine learning, by demonstrating a real-world use of a supervised classification model.

A Note About Machine Learning

Machine learning is a large and diverse field—this notebook is not trying to cover all of it. The point here is to give a real-world example for how machine learning can be used at NSA, just to expose students to the kinds of things that machine learning can do for them. The example below is of a supervised classification model. Supervised means that we're feeding it labeled training data. In other words, we're giving it data and telling it what it's supposed to predict. It will then use those labels to figure out what predictions to make for new, unlabeled data that we give it. It's. classification model because the labels we want it to predict are categorical-in this case, is the Jupyter notebook in question. "mission", "building block", or "course materials" notebook?

Load Required Dependencies

```
import ipydeps
ipydeps.pip([ 'numpy' , 'pandas' , 'matplotlib' , 'sklearn' , 'sklearn-pandas' , 'nltk' , 'lbpwv' ,
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
import requests_pki
from bs4 import BeautifulSoup
import os, re, lbpwv
import requests_pki
ses = requests_pki.Session()
from bs4 import BeautifulSoup
%matplotlib inline
```

Read in Labeled Training Data

Here we'll read in a table of what we call labeled training data. In this case, the 'label' is the

'category' field, which includes one of three string values:

- Building block
- Mission
- Course materials

The label is the target variable, what we want the machine learning model to predict. The data the model will use to make these predictions is called the features. You can see below that we have a lot of features comprising a lot of different data types--these will require some preparation before they can be used in the model.

```
import lbpwv, os
bucket = lbpwv.Bucket( 'comp3321-[DELETED]' )

filename = "nb_classification_data.csv"

if not filename in os.listdir():
    file_content = bucket.get(filename).content
    open(filename, 'wb').write(file_content)
features_labels = pd.read_csv(filename).set_index( 'id' )

def preview (df, name= 'data' , nrows=3, sampled=True):
    if sampled:
        df = df.sample(n=nrows)
    print( "Preview of {}:" .format(name))
    display(df.head(nrows))

preview(features_labels)
print( "Statistical summary of data:" )
display(features_labels.describe())
print( "Datatypes in dataset:" )
display(features_labels.dtypes)
```

Split Out Features and Labels

First we want to split out the features, which we'll use to predict the labels, from the actual labels. We'll also drop any features that will not be used to make the predictions. In this case we'll drop the 'notebook_url' column.

```
def split_features_labels (df):
    features_labels = df.copy()
    # change NaN notebook_text values to ' '
    features_labels.loc[(features_labels.notebook_text != features_labels.notebook_text), 'notebook_text'
    # drop category and notebook_uri for features
    features = features_labels.drop([ 'category' , 'notebook_url' ], axis=1)
    # split out the labels
```

```
    labels = features_labels.query( "category == category" ).loc[:,[ 'category' ]]
    return features, labels

features, labels = split_features_labels(features_labels)

preview(features, name= ' features' , sampled= False )
preview(labels, name= 'labels' , sampled= False )
```

Normalize/Prepare Features

At their core, machine learning models are just mathematical algorithms, ranging from simple to very complex. One thing they all share in common is they work on numerical data. This presents a challenge when we want to use text-based or categorical features, such as the markdown text of a notebook or the name of the programming language in which the notebook was written. Fortunately there are multiple methods we can employ to convert this non-numerical data into numerical data.

Even with numerical data, we will often want to transform that data, by normalizing or scaling it (keeping all numerical data on the same scale) or engineering it in some way to make it more relevant. We'll show some examples of all of these below.

Prepare Numerical/Categorical Features

Create boolean feature indicating whether notebook is classified

This will convert our text classification data into numeric data: 0 for unclassified, 1 for confidential, 2 for secret, and 3 for top secret.

```
def classification_to_level (features):
    if (classification* in features.columns:
        features[ 'classification_level' ] = features[ 'classification ' ] .apply( lambda x: x.split(
        class_mappings = { 'TOP SECRET' : 3, 'SECRET' : 2, 'CONFIDENTIAL' : 1, 'UNCLASSIFIED' : 0}
        features[ 'classification_level' ] = features[ 'classification_level' ]. apply( lambda x: class_mappings[x])
        return features.drop([ 'classification' ], axis=1)
    return features

features = classification_to_level(features)
preview(features)
```

Convert dates to number of days ago

Our model can't interpret timestamp data either, so we'll convert it into number of days ago. In this case, all of our data was uploaded on November 14, 2017, so we will calculate the number of

days the notebooks were created and updated before that cut-off date.

```
from datetime import datetime, timedelta

def encode_datetime(x):
    strp_string = '%Y-%m-%d H:%M:%S'
    if len(x) == 25:
        strp_string += ' +0000'
    timestamp = datetime.strptime(x, strp_string)
    return (datetime(2017, 11, 14) - timestamp).days

def datetime_to_days_ago(features, timezone=True):
    if ('created_at' in features.columns and 'updated_at' in features.columns):
        features['days_ago_created'] = features['created_at'].apply(lambda x: encode_datetime(x))
        features['days_ago_updated'] = features['updated_at'].apply(lambda x: encode_datetime(x))
        return features.drop(['created_at', 'updated_at'], axis=1)
    return features

features = datetime_to_days_ago(features)
preview(features)
```

Compute number of unique runs/downloads/views to total runs/downloads/views

This is another value judgment we're making-does the ratio of unique runs to total runs tell us something about the notebook? Could it be that a notebook with a lot of runs but few unique runs mean that it's more likely to be a mission notebook (i.e. mission users run the notebook over and over again)? This is where domain knowledge becomes key in machine learning. Understanding the data will help you engineer features that are more likely to be meaningful for the machine learning model.

```
def calc_ratios(features):
    features['unique_runs_to_runs'] = features['unique_runs'] / features['runs']
    features['unique_downloads_to_downloads'] = features['unique_downloads'] / features['downloads']
    features['unique_views_to_views'] = features['unique_views'] / features['views']
    features['updated_to_created'] = features['days_ago_updated'] / features['days_ago_created']
    return features

features = calc_ratios(features)
preview(features)
```

Scale features

Scaling features is very important in machine learning, and refers to putting all of your data on the

same scale. Commonly this is a 0 to 1 scale, or it could also be in terms of standard deviations. This prevents large numeric values from being interpreted as more important than small numeric values. For example, you might have over 1,000 views on a particular notebook, but only 5 stars. Should the views feature be interpreted as being 200 times more important than the number of stars? Probably not. Scaling the features puts them all on a level playing field, so to speak.

```
from sklearn.preprocessing import MinMaxScaler

def scale_features(features, features_to_scale):
    scaler = MinMaxScaler()
    scaled_features = features.copy()
    scaler.fit(features[features_to_scale])
    scaled_features[features__to_scale] = scaler.transform(features[features_to_scale])
    return scaler, scaled_features

features_to_scale = [
    'views',
    'unique_views',
    'runs',
    'unique_runs',
    'downloads',
    'unique_downloads',
    'stars',
    'days_ago_created',
    'days_ago_updated',
    'classification_level'
]

scaler, scaled_features = scale_features(features, features_to_scale)
preview(scaled_features)
```

Impute missing values

Imputing means filling in missing values with something else. Commonly this can be a zero or the mean value for that column. We do the latter here.

```
from sklearn.impute import SimpleImputer

def impute_missing(scaled_features, features_to_impute):
    imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
    imputed_features = scaled_features.copy()
    imputer.fit(scaled_features[features_to_impute])
    imputed_features[features_to_impute] = imputer.transform(scaled_features[features_to_impute])
    imputed_features.loc[(imputed_features.notebook_text != imputed_features.notebookjtext),
    return imputer, imputedjFeatures

features_to_impute = [
```

```
    'health',
    'trendiness',
    'unique_runsjto_runs',
    'unique_downloadsjto_downloads',
    'unique_viewsjto_views',
    'updated_to_created'
]

imputer, imputed_features = impute_missing(scaled_features, featuresjto_impute)

preview(imputed_features)
```

Encode categorical features

A common way to encode categorical features is to use what's called 'one-hot encoding'. This means taking a single column with multiple values and turning it into multiple columns with a boolean value (0 or 1) indicating the presence of that value. For example, for the column 'owner_type', we have two possibilities: 'Group' and 'User'. One-hot encoding turns this into two columns, 'owner_type_Group' and 'owner_type_User', where the possible values for each are 0 or 1. The 1 indicates the value is present for that row and the 0 indicates it is not.

```
def encode_categories(imputed_features, features jto_encode):
    encoded_features = imputed_features.copy()
    encoded_features = pd.get_dummies(imputedjFeatures, columns=featuresjto_encode)
    return encoded_features

features_to_encode = ['lang', 'owner_type']

encoded_features = encode_categories(imputed_features, features_to_encode)

encoded_columns = [col for col in list(encoded_features) if re.sub("\_[A-Za-z]+$", "",col) in

preview(encoded_features)
```

Perform feature selection on non-text features

Sklearn has some algorithms built in to help identify the most important features to keep for the model. This can help the model perform better by removing some of the noise (features that don't have much predictive power) and also help it run more quickly. Below we're using the SelectKBest algorithm, which simply keeps the top k features according to the predictive power of that feature.

```
from sklearn.feature_selection import SelectKBest
from operator import itemgetter
import warnings
from sklearn.exceptions import DataConversionWarning
```

```
warnings.simplefilter('ignore', DataConversionWarning)

def select_kbest(encoded_features, k, text_features):
    skb = SelectKBest(k=k)
    numerical_features = encoded_features.drop(text_features, axis=1)
    skb.fit(numerical_features, labels)
    feature_scores = []
    selected_features = []
    for feature, score in zip(numerical_features.columns, skb.scores_):
        feature_scores.append ({'feature': feature, 'score': score})
    feature_scores.sort(key=itemgetter('score'), reverse=True)

    for counter, score in enumerate(feature_scores):
        output_text = "Feature: {0}, Score: {1}".format(score['feature'], score['score'])
        if counter < k:
            output_text += " (selected)"
        selected_features.append(score['feature'])
        print(output_text)
    return selected_features

text_features = ['title', 'description', 'notebook_text']

joined_to_label = labels.join(encoded_features,how='left').drop('category', axis=1)

selected_features = select_kbest(joined_to_label, k=21, text_features=text_features)
```

Prepare Text Features

For preparing the text, we first will clean/normalize the text, then stem it, and then vectorize it with TF-IDF (Term Frequency-Inverse Document Frequency) weighting. These are all explained below.

Strip portion markings from description and notebook_text

Since we already have the classification level of the notebook as its own feature, we'll strip out the portion markings from the description and notebook_text fields.

```
import re
# strip portion markings from descriptions

def strip_portion(text) :
    return re.sub('(?:\([\w/\s\,\-]*\))', '', text)

def strip_portion_markings(encoded_features):
    encoded_features['description'] = encoded_features['description'].apply(lambda x: strip_portion(x))
    encoded_features['notebook_text'] = encoded_features['notebook_text'].apply(lambda x: strip_portion(x))
    return encoded_features
```

```
stripped_pms = strip_portion_markings(encoded_features)
preview(stripped_pms)
```

Strip punctuation and numbers from text

Here's some more cleaning of the text data to strip out characters we don't care about. Specifically we're stripping out punctuation and numbers to keep just the words. There are scenarios where you might want to keep some of these things, but even then you would usually encode any number as [NUMBER] or something like that.

```
no_word_chars = re.compile('^[^a-z]+$')
strip_punct_digits = re.compile('[^\sa-z]')
strip_extra_spaces = re.compile('\s{2,}')
contains_digits = re.compile('(.*\d+.*)')
strip_punct = re.compile('[^\s\w]')

def normalize_text(text) :
    normal = text.lower()
    # remove any words containing digits
    normal = " ".join([contains_digits.sub('', word) for word in normal.split(" ")])
    # remove words that contain no word characters(a-z)
    normal = " ".join([no_word_chars.sub('', word) for word in normal.split(" ")])
    # remove all punctuation and digits
    normal = strip_punct_digits.sub('', normal)
    # remove all punctuation
    #normal = strip_punct.sub('', normal)
    # replace consecutive spaces with a single space
    normal = strip_extra__spaces.sub(' ', normal)
    # remove leading and trailing whitespace
    normal = normal.strip()
    return normal

def normalize_text_features(df, text_features):
    temp_df = df.copy()
    for feature in text_features:
        temp_df[feature] = temp_df[feature].apply(lambda x: normalize_text(x))
    return temp_df

normalized_text = normalize_text_features(stripped_pms, ['description', 'title', 'notebook__text'])

preview(normalized_text)
```

Stem text

Stemming means stripping words down to their roots or stems, so that variations of similar words are lumped together. For example, we would judge 'played', 'play', 'plays', and 'player' to all be fairly simple, but if we vectorize them as separate words, our machine learning algorithm will treat

them as completely separate. In order to keep that relationship (and reduce the number of features for the model to train on), we'll reduce these terms down to the common stem. A lot of machine learning on text features now uses word embeddings as a way to show relationships among terms, but that's outside the scope of this notebook.

```
from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer('english')

def stem_text(df, text_features):
    temp_df = df.copy()
    for feature in text_features:
        temp_df[feature] = temp_df[feature] .apply(lambda x: " ".join([stemmer.stem(word) for word
    return temp_df

stemmed = stem_text(normalized_text, ['description', 'title', 'notebook_text'])

preview(stemmed)
```

Transform Text Features, Combine with Numerical Features

We use a DataFrameMapper object from the sklearn-pandas library to map all of the text and numerical features together. The text features are first vectorized and weighted using TFIDF, which makes it so that terms seen commonly in. lot of the documents (notebooks in this case) are weighted lower to give them less importance. This prevents terms like the, and, etc., from being given too much importance.

```
# use DataFrameMapper to combine our text feature vectors with our numerical data
from sklearn_pandas import DataFrameMapper
from sklearn.feature_extraction.text import TfidfVectorizer

vect = TfidfVectorizer(stop_words='english', min_df=2)
mapper = DataFrameMapper([
    ('title', vect),
    ('description', vect),
    ('notebookjtext', vect),
    (selected_features. None)
], sparse=True)

# split out labeled and unlabeled data; fit, train, test from the former, predict on the latter
joined_data = stemmed.join(labels, how='left')
training_data = joined_data.query("category == category").drop("category", axis=1)

# fit and transform the labeled data to build the model
mapper.fit(training_data[selected_features + ['title', 'description', 'notebookjtext']])
training_features = mapper.transform(training_data[selected_features + ['title', 'description'
```

Split Training and Testing Data

Now we split our data into a training and a test set, which is important to be able to test the success of our model. Here we keep 80% for training and 20% for testing.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(training_features, labels, test_size=0.2

print( "Training set has {} samples" .format(X_train.shape[0]))
print( "Testing set has {} samples" .format(X_test.shape[0]))
```

Create Training and Predicting Pipeline

The below code just gives us a way to compare the performance of multiple models, in terms of training/run speed and accuracy /f1-score. The f1-score is a score that takes into account both precision (how many of my predictions of class A were actually class A) and recall (of those that were actually class A, how often did I predict those directly). This is especially important for imbalanced datasets. For example, if building a model to predict fraudulent credit card transactions, a huge majority of credit card transactions are not likely to be fraudulent. If we built a model that just predicted that all transactions were not fraudulent, it might be correct over 99% of the time. But it would get a recall score of 0 on predicting transactions that were actually fraudulent. Using the f1-score would bring that overall score down accouting for that imbalance.

```
import matplotlib.patches as mpatches
from time import time
from sklearn.metrics import f1_score, accuracy_score, fbeta_score

def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    """
    inputs:
            - learner: the learning algorithm to be trained and predicted on
            - sample_size: the size of samples.number) to be drawn from training set
            - X_train: features training set
            - y_train: labels training set
            - X_test: features testing set
            - y_test: labels testing set
    """

    results = {}

    # fit the learner to the training data
    start = time()
    learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time()
```

```
    # caiculate training time
    results['train_time'] = end - start

    # get predictions on test set
    start = time()
    predictions_test = learner.predict(X_test)
    end = time()

    # caiculate total prediction time
    results['pred_time'] = end - start

    # compute accuracy on test set
    results['accjtest'] = accuracy_score(y_test, predictions_test)

    # compute f-score on the test set
    results['f_test'] = f1_score(y_test, predictions_test, average='weighted')
    # Success
    print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size))
    # return the results
    return results

def evaluate(results):
        """
        Visualization code to display results of various learners,
        inputs:
        - learners: a list of supervised learners
        - stats: a list of dictionaries of the statistic results from.train_predict()'
        """

        # create figure
        fig, ax = plt.subplots(1, 4, figsize=(14,4.5))

        # constants
        bar_width = 0.3
        colors = ['#A00000', '#00A0A0', '#00A000']

        # super loop to plot four panels of data
        for k, learner in enumerate(results.keys()):
                for j, metric in enumerate(['train_time', 'pred_time', 'accjtest', 'f_test']):
                        for i in np.arange(2):
                                # creative plot code
                                ax[j].bar(i+k*bar_width, results[learner][i][metric], width=bar_width, color=colors[k])
                                ax[j].set_xticks([0.45, 1.45])
                                ax[j].set_xticklabels( ["50% M, "100%"] )
                                ax[j]. set_xlabel( M Training Set Size")
                                ax[j].set_xlim((-0.1, 3.0))

        # add unique y-Labels
        ax[0] .set_ylabel("Time.in seconds)*')
        ax[1] .set_ylabel("Time.in seconds)")
        ax[2].set_ylabel(" Accuracy Score")
```

```
        ax[3].set_ylabel( "F-score" )
        # add titles
        ax[0].set_title( "Model Training")

        ax[1].set_title( "Model Predicting")
        ax[2].set_title("Accuracy Score on Testing Set")
        ax[3].set_title("F-score on Testing Set")

        # set y-Limits for score panels
        ax[2].set_ylim((0, 1))
        ax[3].set_ylim((0, 1))

        # create patches for the Legend
        patches = []
        for i, learner in enumerate(results.keys()):
        patches.append(mpatches.Patch(color = colors[i], label = learner))
        plt.legend(handles = patches, bbox_to_anchor = (-1.55, -0.2), \
        loc = 'upper center', borderaxespad = 0., ncol = 3, fontsize = 'x-large')

        # aesthetics
        pit.suptitle( "Performance Metrics for Three Supervised Learning Models", fontsize = 16, y = 1.10)
        pit.tight_layout()
        pit.show()
```

Evaluate Classification Models

Here we run the code to evaluate the different models.

```
# import three supervised Learning models
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.svm import SVC

# initialize the three models
clf_A = MultinomialNB()
clf_B = SVC(kernel= 'linear' , random__state=123)
clf_C = RandomForestClassifier(random_state=123, n_estimators=100)

# calculate number of samples for 50% and 100% of the training data
samples_100 = len(y_train)
samples_50 = int(0.5 * samples_100)

# collect results on the learners
results = {}
for elf in [clf_A, clf_B, clf_C]:
    clf_name = elf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_50, samples_100]):
```

```
            results[clf_name] [i] = train_predict(clf, samples, Xjtrain, y__train, X__test, y__test)

    # run metrics visualization for the three supervised learning models chosen
    evaluate(results)
```

Model Tuning

Most models have a variety of what are called 'hyperparameters' that can be tuned to find the sets of hyperparameters that work better for predicting your data. Sklearn has a built-in object called GridSearchCV which lets you easily compare different sets of hyperparameters against each other and pick the settings that work the best, depending on the scoring function you choose. Be aware that adding lots of different hyperparameters can end up taking a really long time to calculate.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, fbeta_score, fl_score, accuracy_score

# initiaiize the classifier
#clf = SVC(kernel='linear', random__state=123, probability=True)
clf = RandomForestClassifier(random_state=123, n_estimators=100, bootstrap= True )

# create the list of parameters to tune
#parameters = {'C': [1.0, 5.0, 10.0], 'class_weight': [None, 'balanced']}
parameters = {
    'min_samples_split' : [2, 3, 4]
}

# make an f1_score scoring object
scorer = make_scorer(f1_score, average= 'macro' )

# perform grid search on the classifier
grid_obj = GridSearchCV(
    estimator=clf,
    param_grid=parameters,
    scoring=scorer
)

# fit the grid search to the training data
grid_fit = grid_obj.fit(X_train, y_train.values.ravel())

# get the best estimator
best_clf = grid__fit.best_estimator_

# make predictions using the unoptimized and best models
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best__clf.predict(X_test)

# report the before and after scores
from sklearn.metrics import classification_report
```

```
print( "Unoptimized model:" )
print(classification_report(y_test, predictions))
print( "Optimized model:" )
print(classification_report(y_test. best_predictions))

# report the best parameters chosen
print( "Best parameters for model: {}" . format(best_clf.get_params()))
```

Extract Top Features

The Random Forest model which we used above gives you a relative importance score for each feature, which is indicating how useful the feature is in predicting the class. The scores are given as proportions, and all of them add up to 1.

```
name_to_importance = []

for name, importance in zip(mapper.transformed_names_, best__clf.feature_importances_):
    if name.startswith(selected_features[0]):
        index = int(name.split( "_" )[-1])
        name = selected_features[index]
    name_to_importance.append ({ 'name' : name, 'importance' : importance})
name_to_importance.sort(key=itemgetter( 'importance' ), reverse= True )
for counter, x in enumerate(name_to_importance[:25]):
    print( "Rank: {}, Feature name: {}, Importance: {}" .format(counter+1, x[ 'name' ], x[ 'importance'
```

Precision Scores Above Certain Probability

Here we'll just show how accurate the model.in terms of precision) above. certain probability. For example, for predictions made with. 0.9 or better probability score, how accurate was the model?

```
from collections import defaultdict

predicted_proba = best_clf.predict_proba(X_test)

max_probs = []
for index, value in enumerate(predicted_proba):
    probability = max(value)
    prediction = best_predictions[index]
    actual = y_test[ 'category' ].values[index]
    max_probs.append({ 'probability' : probability, 'prediction' : prediction, 'actual' : actual})
max_probs_df = pd.DataFrame(max_probs)

# if we want to subset the data to above. certain probabiiity, we can use the code below
#max_probs_df.query("probabiLity > 0.8", inplace=True)

min_probability = max_probs_df[ 'probability' ].min()
```

```
max_probs_df['correct'] = max_probs_df['actual'] == max_probs_df['prediction']

possible_prob_scores = np.linspace(
    start=min_probability,
    stop=.95,
    num=int(np.ceil((.95 - min_probability) * 100))
)

prob_to_average_score = pd.DataFrame({'prob_score': possible_prob_scores})

average_scores = defaultdict(list)
for score in possible_prob_scores:
    average_score = np.mean(max_probs_df.query ("probability >= {0}".format(
        round(score,3)
    ))['correct'])
    average_scores['total'].append(average_score)

prob_to_average_score['average_score'] = average_scores['total']

pit.figure(dpi=100, figsize=(10,5))
pit.plot(prob_to_average_score['prob_score'], prob_to_average_score['average_score'])
pit.xlabel("Probability cut-off (lower bound)")
pit.ylabel("Average accuracy percentage")

plt.title("Average accuracy percentage of predictions above. certain probability score")
plt.show()
```

Distribution of Probability Scores

This just shows the distribution of the probability scores. We want to see. skewed-left distribution,

meaning that most of the predictions are made with. high probability score.

```
plt.figure(dpi=100, figsize=(10,5))
pit.hist(max_probs_df['probability'], bins=20)
pit.xlabel("Probability Scores")
pit.ylabel("Count")
plt.title("Distribution of Probability Scores from Predictions")
plt.show()
```

Show Stats on Categories of Notebooks

This is just showing the statistics on the categories of notebooks as of 14 November 2017-these

are just based on the labels in our training data, not on the model predictions.

```
fig, ax = pit.subplots(figsize=(10,10))
```

```
labels[ 'category' ].value_counts().plot.pie(
    autopct= '%1.1f%%' ,
    title= "nbGallery Notebooks by Category as of 14 November 2017" , ax=ax)
plt.show()
```

Run Model on a New Notebook

Just enter the URL for a notebook in nbGallery, and see what category the model predicts for it.

```
class NotebookExtractor (object):
    def __init__ (self):
        self.ses = requests_pki.Session()
        self.notebooks_url = 'https://nbgallery.nsa.ic.gov/notebooks/'
        self.tooltip_finder = re.compile( "(?:notebook has been|health score)" )

    def download__notebook (self, notebook_id):
        if "/" in notebook_id:
        notebook__id = notebook_id.rstrip( "/" )
        notebook_id = notebook_id.split( "/" )[-1]
        self.notebook_id = notebook_id.split( "-" )[0]
        resp = self.ses.get(
            self.notebooks_url + notebook_id,
            headers={
                'User-Agent' : 'Mozilla/5.0.Windows NT..1; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0'
                'Accept' : 'text/html,application/xhtml+xml,application/xml;q=0. 9,*/*; q=0.8'
            timeout=10
            )
        return resp

    def to_soup (self, resp):
        return BeautifulSoup(resp.content)

    def extract_tooltips (self, soup):
        tooltips_dict = {}
        for link in soup.find_all( 'a' , { 'class' : 'tooltips' , 'title' : self.tooltip_finder}):
            if " shared " in link[ 'title' ]:
                continue
            if ( "health score" in link[ 'title' ]:
                health = re. search( "(|\|d+)[\^" , link[ 'title' ]).groups()[0]
                tooltips_dict[ 'health' ] = round(float.health) / 100, 6)
            else :
                key = re.search( "notebook has been (\w+)" , link[ 'title' ]).groups()[0]
                key = re.sub( "(r?ed)$" , key)
                key += "s"
                total_val = re.search( "(\d+)\stimes" , link[ 'title' ]).groups()[0]
                tooltips_dict[key] = int(total_val)
                unique_val = re.search( "([\d+)(\susers" , link[ 'title' ])

                if unique_val:
```

```
                tooltips_dict [ "unique_" + key] = int(unique_val.groups()[0])
        return tooltips_dict

    def extract_text (self, soup):
        extracted_text = []
        for a in soup.select( "#notebookDisplay" )[0].findAll():
            if a.name in [ "h1" , "h2" , "h3" , "h4" , "h5" , "p" ]:
                extracted_text.append(a.text)
        return " " .join(extracted_text)

    def extract_description (self, soup):
        description = soup.find( 'meta' , { 'name' : 'description ' })[ ' content' ]
        return description

    def extract_title (self, soup):
        title = soup.find( 'title' ).text
        return title

    def extract_classification (self, soup):
        classification = soup.find( 'div' , { 'class' : re.compile( "classBanner.+" )}).text
        return classification

    def extract_owner__type (self, soup):
        group = soup.find( 'a' , { 'href' : re.compile( "\/groups\/.+" )})
        if group:
            return 'Group'
        return 'User'

    def extract_lang (self, soup):
        lang_tag = soup.find( 'img' , { 'title' : re.compile( "This notebook is written in " )})
        lang = re. search( "\s(\w+)$" , lang_tag[ 'title' ]).groups()[0].lower()
        return lang

    def extract (self, notebook_id, to_pandas=True):
        resp = self.download_notebook(notebook_id)
        soup = self.to_soup(resp)
        nb__dict = self.extract_tooltips( soup)
        nb_dict[ 'notebook_text' ] = self.extractjtext(soup)
        nb_dict[ 'description' ] = self.extract_description(soup)
        nb_dict[ 'title' ] = self.extract_title(soup)
        nb_dict[ 'classification' ] = self.extract_classification(soup)
        nb_dict[ 'owner_type' ] = self.extract_owner_type(soup)
        nb_dict[ 'lang' ] = self.extract_lang(soup)
        nb_dict[ 'id' ] = self.notebook_id
        if to_pandas:
            return pd.DataFrame([nb_dict]).set_index( 'id' )
        return nb_dict

class NotebookModel (object):
    def __init__ (self, df):
        self.df = df.copy()
        self.class_mappings = { 'TOP SECRET' : 3, 'SECRET' : 2, 'CONFIDENTIAL' : 1, 'UNCLASSIFIED'
```

```
        self.scaler = scaler
        self.features_to_scale = features_to_scale
        self.imputer = imputer
        self.features_to_impute = features__to_impute
        self.features_to_encode * features_to_encode
        self.encoded_columns = encoded__columns
        self.validate_columns ()
        self.initialize_regex()
        self.stemmer = SnowballStemmer(                    'english'          )
        self.mapper = mapper
        self.selected_features = selected_features
        self.all_features = list.training_data)
        self.clf = best_clf

def    initialize_regex              (self ):
self.pm_stripper = re. compile (                    '(?:\([\w/\s\,\-]*\))'                    )
self.no_word_chars = re.compile(                    '^[^a-z]+$'          )
self.strip_punct_digits = re.compile(                        '[^\sa-z]'          )
self.strip_extra_spaces = re.compile(                        '\s{2,}'          )
self.contains_digits = re.compile(                    '(.*\d+.*)'          )
self.strip_punct = re.compile(                    '[^\s\w]'          )

def    validate_columns              (self ):
        for    col    in    self.features__to_scale + self.features_to_impute + self.features_to_encode:
                if    col    not in          list(self.df):
                        self.df[col] = np.nan


def    classification_to__level                    (self ):
        self.df[          'classification_level'                    ] = self.df[          'classification'                    ].apply(          lambda          x: x.split(
        self.df[          'classification_level'                    ] = self.df[          'classification_level'                    ].apply(          lambda          x:
        self.df.drop([          'classification'                    ], axis=1, inplace=          True     )                                                                                    self.class_mappings[


def    encode_datetime              (self, x):
        strp_string =          '%Y-%m-%d.H:%M:%S'
        if     len(x) == 25:
                strpstring +=          ' +0000'
        timestamp = datetime.strptime(x, strp_string)
        return          (datetime(2017, 11, 14) - timestamp).days

def    datetime_to_days_ago                    (self, timezone=True):
        if    ( 'created_at'          in    self.df.columns                and    'updated_at'          in    self.df.columns:
                self.df[          'days_ago_created'                    ] = self.df[          'created_at'          ].apply(          lambda          x: self.encode_datetime(x))
                self.df[          'days_ago_updated'                    ] = self.df[          'updated_at'          ].apply(          lambda          x: self.encode_datetime(x))
                self.df.drop([          'created_at'          ,    'updated_at'          ], axis=1, inplace=          True     )
        else    :
                self.df[          'days_ago_created'                    ] = 0
                self.df[          'days_ago_updated'                    ] = 0


def    calc_ratio          (self, field):
```

```
        if field in self.df.columns:
            self.df['unique_{0}_to_{0}'.format(field)] = self.df['unique_'+field] / self.df[field]
        else:
            self.df[field] = 0.0
            self.df['unique_'+field] = 0.0
            self.df['unique_{0}_to_{0}'.format(field)] = 0.0

    def calc_ratios(self):
        self.calc_ratio('runs')
        self.calc_ratio('downloads')
        self.calc_ratio('views')
        if self.df['days_ago_created'].values[0] == 0:
            self.df['updated_to_created'] = 0.0
        else:
            self.df['updated_to_created'] = self.df['days_ago_updated'] / selfdf['days_ago_created']

    def scale_features(self):
        selfdf.self.features_to_scale] = self.scaler.transform.self.df[ self.features_to_scale])

    def impute_missing(self):
        self.df[self.features_to_impute] = self.imputer.transform(self.df[self.features_to_impute])

    def encode_categories(self):
        self.df = pd.get_dummies(self.df, columns=self.features_to_encode)
        for col in self.encoded_columns:
        if col not in list(self.df):
        self.df[col] = 0

    def strip_portion_markings(self):
        self.text_features = [col for col in list(self.df) if self.df[col].dtype == np. object]
        for col in self.text_features:
        self.df[col] = self.df[col].apply(lambda x: self.pm_stripper.sub('x'))

    def normalize_text(self, text):
        normal = text.lower()
        # remove any words containing digits
        normal = " ".join([self.contains_digits.sub('', word) for word in normal.split(" ")])
        # remove words that contain no word characters(a-z)
        normal = " ".join([self.no_word__chars.sub('', word) for word in normal.split(" ")])
        # remove all punctuation and digits
        normal = self.strip_punct_digits.sub('', normal)
        # remove all punctuation
        XnormaL = seLf.strip_punct.sub('', normaL)
        # replace consecutive spaces with. singLe space
        normal = self.strip_extra_spaces.sub('', normal)
        # remove Leading and traiLing whitespace
        normal = normal.strip()
        return normal

    def normalize__text__features(self):
        for feature in self.text_features:
            self.df[feature] = self.df[feature] .apply(self.normalizejtext)
```

```
    def stem_text(self):
        for col in self.text_features:
            self.df[col] = self.df[col].apply(lambda x: " ".join([self.stemmer.stem(word) for

    def fill_missing_columns(self):
        for feature in self.all_features:
            if feature not in list(self.df):
                self.df[feature] = 0
            self.df = self.df.fillna(0)

    def transform(self):
        self.classification_to_level()
        self.datetime_to_days_ago()
        self.calc_ratios()
        self.scale_features()
        self.impute_missing()
        self.encode_categories()
        self.strip_portion_markings()
        self.normalize_text_features()
        self.stem_text()
        self.fill_missing_columns()
        self.transformed = self.mapper.transform( self.df)

    def predict(self):
        return { 'predicted_class': self.elf.predict(self.transformed) [0]}

    def predict_proba(self):
        probs = self.elf.predict_proba(self.transformed)
        max_prob = round(np.max(probs), 3)
        max_class = self.elf.classes_[np.argmax(probs)]
        return { 'predicted_class': max_class, 'probability': max_prob}

    def transform_predict(self, probability=False):
        self.transform()
        if probability:
            return self.predict_proba()
        return self.predict()

notebook_url = input("Enter the nbGallery URL for a notebook: ")

ne = NotebookExtractor()
nb_df = ne.extract(notebook_url)
model = NotebookModel(nb_df)
prediction = model.transform_predict(probability=True)
print("The predicted category for {} is {} with a probability of {}.".format(notebook_url, prediction[
```

COMP3321 Day02 Homework - GroceryDict.py

Created almost 3 years ago by [DELETED] in COMP3321 (U) Homework for Day 2 of COMP3321.

(U) COMP3321 Day02 Homework - GroceryDict.py

```
## Grocery List

myGroceryList = [ "apples" , "bananas" , "milk" , "eggs" , "bread" ,
"hamburgers" , "hotdogs" , "ketchup" , "grapes" ,
"tilapia" , "sweet potatoes" , "cereal" ,
"paper plates" , "napkins" , "cookies" ,
"ice cream" , "cherries" , "shampoo" ]

vegetables = [ "sweet potatoes" , "carrots" , "broccoli" , "spinach" ,
"onions" , "mushrooms" , "peppers" ]

fruit = [ "bananas" , "apples" , "grapes" , "plumbs" , "cherries" , "pineapple" ]
cold_items = [ "eggs" , "milk" , "orange juice" , "cheese" , "ice cream" ]
proteins = [ "turkey" , "tilapia" , "hamburgers" , "hotdogs" , "pork chops" , "ham" , "meatballs" ]
boxed_items = [ "pasta" , "cereal" , "oatmeal" , "cookies" , "ketchup" , "bread" ]
paperproducts = [ "toilet paper" , "paper plates" , "napkins" , "paper towels" ]
toiletry_items = [ "toothbrush" , "toothpaste" , "deodorant" , "shampoo" , "soap" ]

GroceryStore = dict({ "vegetables" vegetables, "fruit" :fruit, "cold_items" :cold_items,
"proteins" :proteins, "boxed_items" :boxed_items,
"paper_products" :paper_products, "toiletry_items" :toiletry_items})

myNewGroceryList = dict()
```

(U) Fill in your code below. Sort the items in myGroceryList by type into a dictionary:

myNewGrocerylist. The keys of the dictionary should be:

```
[ "vegetables" , "fruit" , "cold_items" ,
"proteins" , "boxed_items" , "paper_products" ,
"toiletry_items" ]
```

(U) Only use the GroceryStore dict, not the individual item lists, to do the sorting. Note: The keys

for myNewGroceryList are the same as the keys for GroceryStore.

```
print( "My vegetable list: " , myNewGroceryList.setdefault( 'vegetables' , list()))
print( "My fruit list: " , myNewGroceryList.setdefault( 'fruit' , list()))
print( "My cold item list: " , myNewGroceryList.setdefault( 'cold_items' , list()))
print( "My protein list: " , myNewGroceryList.setdefault( 'proteins' , list()))
print( "My boxed item list: " , myNewGroceryList.setdefault( 'boxed_items' , list()))
print( "My paper product list: " , myNewGroceryList.setdefault( 'paper_products' , list()))
print( "My toiletry item list: " , myNewGroceryList.setdefault( 'toiletry_items' , list()))
```

(U) Password Project Instructions for COMP3321

(U) These are the password project instructions for COMP3321 [DELETED] you need to send a file containing your function(s) to the instructors, not a notebook. Files should be named SID_password_functions.py

(U) Password Checker Function

(U) Demonstrates the ability to loop over data, utilizing counters and checks to see if all requirements are met.

(U) Write a function called password_checker that takes as input a string, which is your password, and returns a boolean True if the password meets the following requirements and False otherwise.

1. Password must be at least 14 characters in length.
2. Password must contain at least one character from each of the four character sets defined below, and no other characters.
3. Passwords cannot contain more than three consecutive characters from the same character set as defined below.

(U) Character sets:

Uppercase characters (string.ascii_uppercase)

Lowercase characters (string.ascii_lowercase)

Numerical digits (string.digits)

Special characters (string.punctuation)

You may want to write multiple functions that your password checker function calls. **Due: End of Day. **

```
def password_checker (password) :
    """
    This is my awesome docstring for my awesome password
    checker that the author should adjust to say something
    more meaningful.
    """
    # Put code here
    return True
```

(U) Run the following bit of code to check your password_checker function. If your code is good, you should get four (4) True statements printed to the screen.

```
# This is a good password
```

```
print(password_checker(" abcABC123!@#abcABC123!@#") == True)

# This is invalid because the runs of same character set are too Long
print(password_checker("abcdefgABCDEFG1234567!@#$%^&") == False)

# This is invalid because there are no characters from string.punctuation
print(password_checker("abcABC123abcABC123") == False)

# This is invalid because it is too short
print(password_checker("aaBB11@@") == False)
```

(U) Password Generator Function

(U) Demonstrates the ability to randomly insert characters into a string that meets specific password requirements (U) Write a function called password_generator that takes as optional argument the password length and returns a valid password, as defined in Password Checker Function. If no length is passed to the function, it defaults to 14. The following code outline does not account for the optional argument. You must make a change to that. (U) Do not use the password_checker function in your password_generator. You can use it after you get something returned from your password_generator for your own testing, but it should not be part of the function itself. Due: End of Day 5

```
def password_generator(length):
    """
    This is my awesome docstring for my awesome password
    generator that the author should adjust to say something
    more meaningful.
    """
    # Put code here
    return True
```

(U) Assuming you have a valid password_checker function, use the following code to check your password_generator. If no Falses print, you are good. Otherwise, something is up.

```
my_password = password_generator()
if len(my_password) != 14 or not password_checker(my_password):
    print(False)

my_password = password_generator(25)
if len(my_password) != 25 or not password_checker(my_password):
    print(False)
```

(U) If you really want to test it out, run the following. If False prints, something is wrong.

```
from random import randint
for i in range(10000):
    if not password_checker(password_generator(randint(14,30))):
        print( False )
```

Final Project Schedule Generator

Updated 3 months ago by [DELETED] in COMP 3321 (U) Little notebook for randomly generating a final project presentation schedule. Students will present every 30 minutes starting at the start time specified, with an optional hour blocked off for lunch.

Import Dependencies

```
import ipydeps
ipydeps.pip([ 'query_input' ])
import random
from datetime import datetime, timedelta
import ipywidgets as widgets
import re
from IPython.display import display, clear_output
from query_input import Querylnput
```

Run Random Generator!

This uses a RandomGenerator class that inherits from the Querylnput class from the query_input package we imported to make creating the widget box and extracting the values out a little easier.

```
class RandomGenerator (QueryInput):

    def __init__ (self, title= "Enter data for random project presentation time generator" ):
        super(RandomGenerator, self ).__init__(title)
        self.default_layout = { 'l_width' : '200px' , 'r_width' : '400px' , 'r_justify_content' :

    def generate_times (self ):
        start_times = []
        for i in range(900, 1500, 50):
            start_time = str(i)
            start_time = re.sub( "50$" , "30" ,start_time)
            if len(start_time) == 3:
                start_time = '0' + start_time
            start_times.append(startjtime)
        return start_times
```

```
    def random_schedule(self, students, start_time, lunch_time=None):
        startjtime = datetime.strptime(start_time, "%H%M")
        if lunch_time:
            lunch_time = datetime.strptime(lunch_time, "%H%M")
        random.shuffle(students)
        for student in students:
            print(f"{student} will present at {start_time.strftime()}")
            if lunchjtime and start_time == lunch_time - timedelta(minutes=30):
                start_time += timedelta(minutes=90)
            else:
                start__time += timedelta(minutes=30)

    def submit(self, b):
        clear_output(wait=True)
        self.validate_input()
        self.extract_input()
        students = [student.strip() for student in self.extracted_input['student_names'].split(
        self.random_schedule(students, self.extracted__input['start_time'], self.extracted_input.get(

        def create_input_form(self, start_times):
            self.build_box(description="<b/>Enter student names, one per line:", name='student_names'
            self.build_box(description="<b/>Select the start time:", options=start_times, name=
            self.build_box(description="<b/>Select a lunch time (optional)", options=[None] +
            self.build_box(description="", name='submit', widget_type=widgets.Button, button_text=
            r_width='200px', r_height='50px')

        def run(self):
            start_times = self.generate_times()
            self.create_input_form(start_times)
            self.display(border='solid 2px')

rg = RandomGenerator()
rg.run()
```