ANLY560 FUNCTIONAL PROGRAMMING FOR ANALYTICS

UNIT 1 – COURSE INTRODUCTION

FEYZI BAGIROV

LESSON OUTCOMES

- Understand course expectations
- Understand overall flow of semester
- Get to know your professor
- Get to know your classmates
- Describe key programming paradigms such as imperative, functional, data-driven, and object-oriented
- Explain the advantages and disadvantages of scripted and compiled languages
- Describe each of the 3 control structures: sequence, selection and interaction
- Identify control structures in various programming languages
- Discuss key aspects of structured, unstructured, and tidy data

COURSE DESCRIPTION

- This course provides the student with the required knowledge and skills to handle and analyze data using a variety of programming languages as well as a variety of programming tools and methods. Depending on current industry standards, the student will be provided with the opportunity to develop knowledge and skills in programming environments such as Java, SQL, and Python. In addition, the student is introduced to current industry standard data analysis packages and tools such as SQL Workbench, NumPy and Pandas, and the Eclipse IDE for Java.

COURSE EXPECTATIONS

- Read the Syllabus
- Introduce Yourself in the Icebreaker forum
- Live Sessions
- Grading
  - Lab Assignments
  - 2 Exams
  - Participation
- Getting Help
  - Leveraging Community
  - Asking Questions

CITING RESOURCES

- APA style citations are the preferred format for crediting your sources
- Both in-text citations and reference pages are required
- Refer to http://www.apastyle.org if you need a review of the proper format

Relational database systems do not lend themselves well to clustering computing ; even though some RDBMS offerings have clustering component, such as Oracle Real Application Cluster (RAC), they still rely on big hardware and a shared storage array. (Sadalage & Fowler, 2013)

Sadalage, R. J., & Fowler, M. (2013). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River, NJ: Pearson Education, Inc.

ACADEMIC HONESTY

- Academic integrity is the pursuit of scholarly activity free from fraud and deception, and is the educational objective of the institution.
- Academic dishonesty includes, but is not limited to
  - Cheating
  - Plagiarism
  - Fabrication of information or citations
  - Facilitating acts of academic dishonesty by others
  - Unauthorized possession of examinations
  - Submitting work of another person, or work previously used without informing the instructor
  - Tampering with the academic work of other students.

FLOW OF THE SEMESTER

- Programming concepts
- Roughly 4 blocks based on programming paradigms
- Languages
- Control structures
- Data
- Computing environment
- Hardware & Software
- Software version control

FLOW OF THE SEMESTER (CONT'D)

- Paradigm discussion
  - Imperative
  - Object oriented
  - Declarative
  - Functional
- Distinctive features
- General applications
- Data Applications
- Wrap-up
- 3 to 4 labs
- 2 exams

A LITTLE ABOUT ME…

Education

- BS – Economics – Azerbaijan University – Baku, Azerbaijan
- MBA - Babson College – Wellesley, Massachusetts
- PhD – Data Sciences– Harrisburg University of Science and Technologies- Harrisburg, Pennsylvania

Teaching

- University of Maryland University College – Data Analytics
- Southern New Hampshire University – Data Analytics
- Becker College – Data Science
- Harrisburg University of Science and Technology – Data Analytics

Research

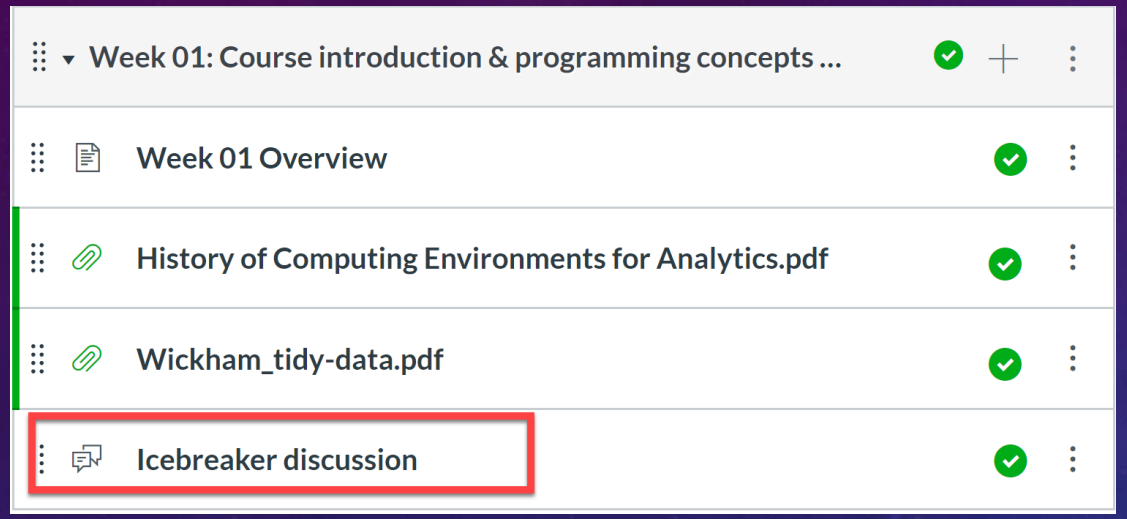
- Natural Language Processing
- Blockchain

A LITTLE ABOUT ME…

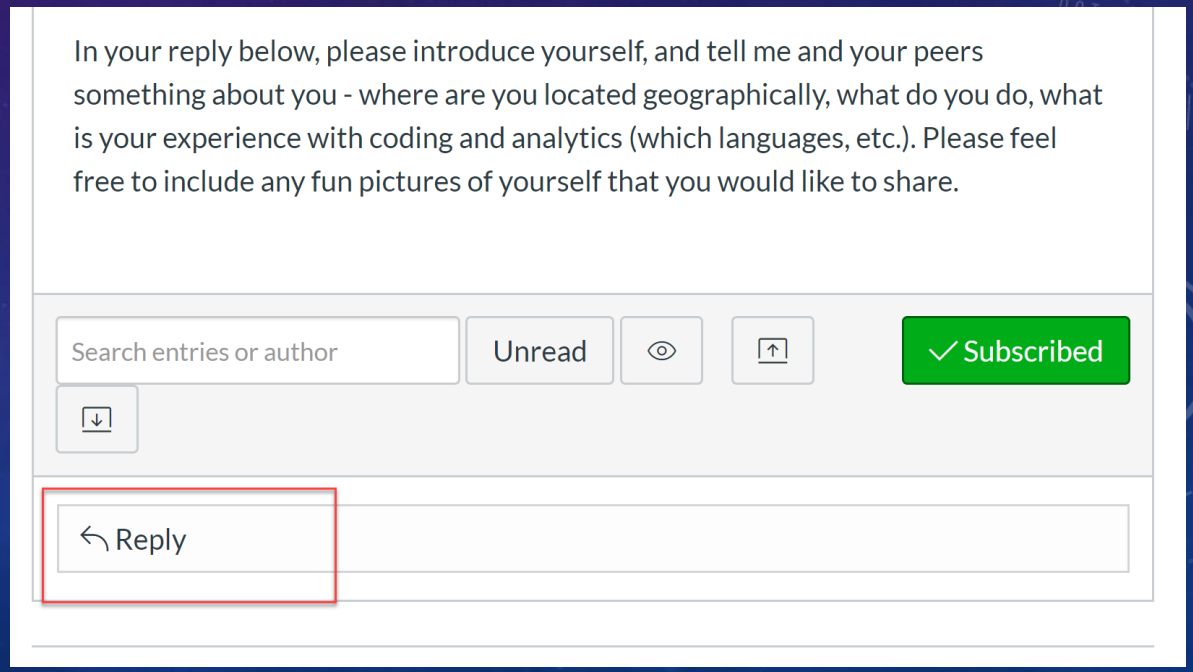
Where have I lived?

- Baku, Azerbaijan
- Odessa, Ukraine
- Rockville, Maryland
- Arlington, Virginia
- Charlotte, North Carolina
- Boston, Massachusetts
- San Diego, California
- West Hartford, Connecticut

GET TO KNOW YOUR CLASSMATES….



- Name/Nickname
- Where do you currently live?
- What is your profession?
- What is your experience with Analytics and coding?
- What made you decide to study Data Analytics?

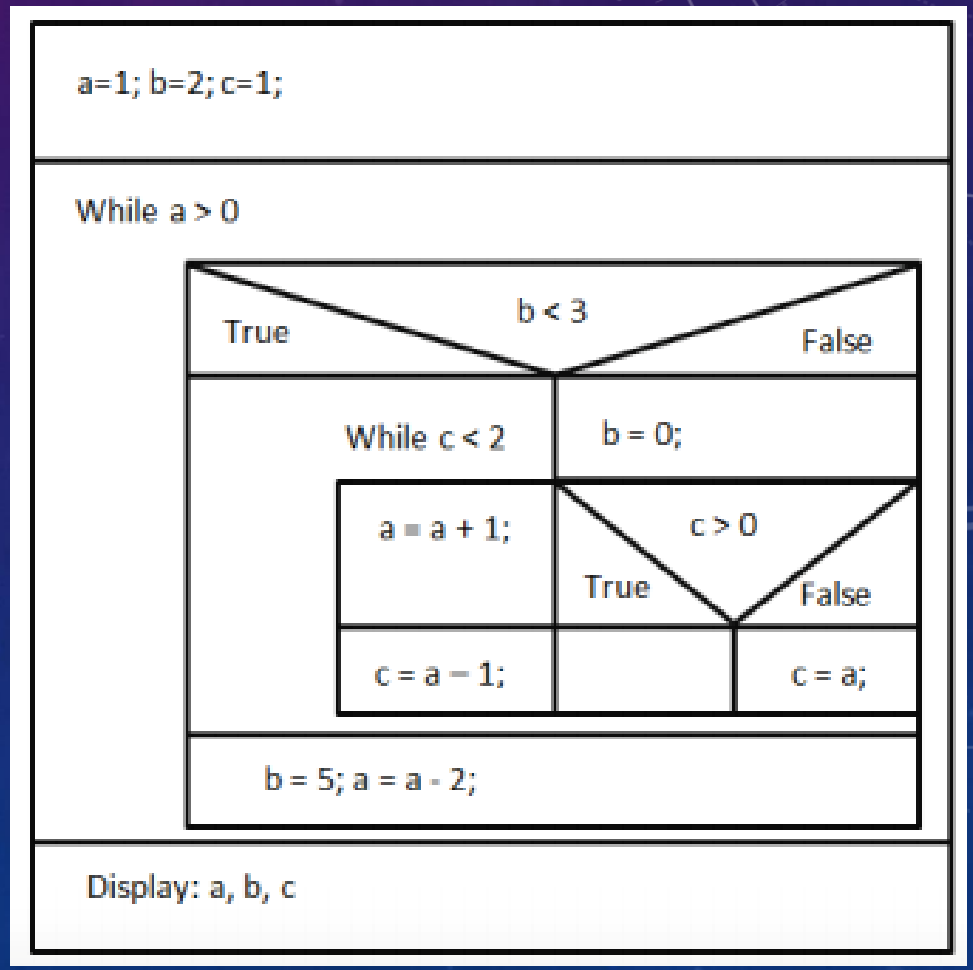


PROGRAMMING PARADIGMS

- A way to classify programming languages based on their features. Languages can be classified into multiple paradigms
- Some paradigms are concerned mainly with implications for the execution model of the language, such as allowing side effects, or whether the sequence of operations is defined by the execution model.
- Other paradigms are concerned mainly with the way that code is organized, such as grouping a code into units along with the state that is modified by the code.
- Yet others are concerned mainly with the style of syntax and grammar.

PROGRAMMING PARADIGMS - IMPERATIVE

- Programs as statements that directly change computed state (data fields)
- A style of programming with a more logical, stepwise program structure
- Derived from structured programming, based on the concept of modular programming of the procedure call
- Main traits
  - Direct assignments, common data structures, global variables
  - Structograms, indentations, no or limited use of GOTO statements
  - Local variables, sequence, selection, iteration, and modularization
- Examples –C, C++, Java, Python



PROGRAMMING PARADIGMS – OBJECT ORIENTED

- Treats data as object manipulated through predefined methods only
- Main traits
  - Objects with methods
  - Message passing
  - Information hiding, data abstraction, encapsulation
  - Polymorphism and inheritance
  - Serialization and marshalling
- Examples – Common Lisp, C++, Effel, Java, PHP, Python, Ruby, Scala

PROGRAMMING PARADIGMS - DECLARATIVE

- Defines program logic, but not detailed control flow
- Main uses
  - Fourth-generation languages
  - Spreadsheets
  - Report program generators
- Examples – SQL, Regular Expressions (RegEx), CSS, Prolog, OWL, SPARQL

PROGRAMMING PARADIGMS - FUNCTIONAL

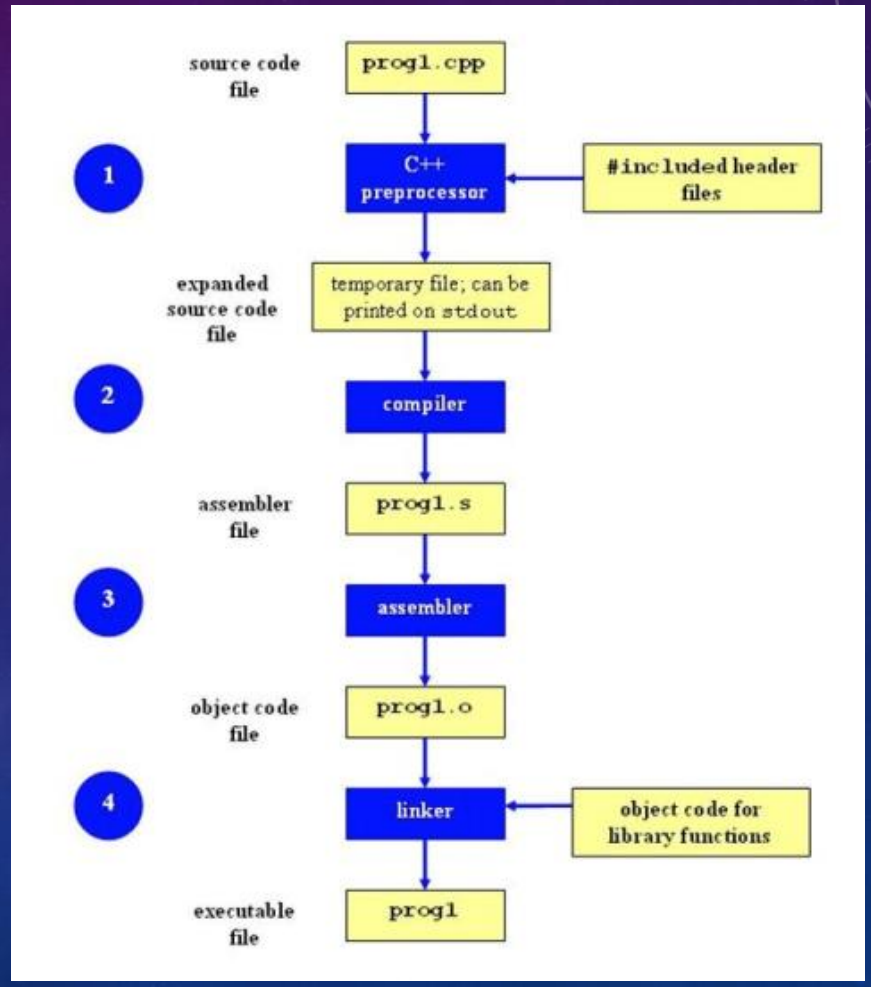
- Treats computation as the evaluation of mathematical function, avoiding state and mutable data
- Main traits:
  - Lambda calculus
  - Compositionality
  - Recursion
  - Referential transparency
  - No side effects
- Examples – C++, Haskell, Lisp, Python, Ruby, Scala, Standard ML, JavaScript, R

SCRIPTED VS. COMPILED LANGUAGES

- Scripted or Interpreted language
  - An interpreted language is a programming language for which most of its implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions.
  - The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code.
- Compiled language
  - Program Instructions (“source code”) are interpreted by a compiler program, but rather than executing the statements, the compiler creates a machine-specific representation of the entire program that can be executed at a later time.
- Hybrid language
  - Some languages exhibit characteristics of both interpreted and compiled languages, with a compilation process that creates a portable set of instructions that are later run by an interpreter process.

SCRIPTED VS COMPILED LANGUAGES

- Compilation process
  1. Preprocessing
  2. Compilation
  3. Assembly
  4. Linking



SCRIPTED VS COMPILED LANGUAGES

- PREPROCESSING

```
cc -E hello_world.c
```

```c
/*
 * "Hello, World!": A classic.
 */

#include <stdio.h>

int
main(void)
{
        puts("Hello, World!");
        return 0;
}
```

```
[lines omitted for brevity]

extern int __vsnprintf_chk (char * restrict, size_t,
       int, size_t, const char * restrict, va_list);
# 493 "/usr/include/stdio.h" 2 3 4
# 2 "hello_world.c" 2

int
main(void) {
 puts("Hello, World!");
 return 0;
}
```

SCRIPTED VS COMPILED LANGUAGES

- COMPILATION

```
cc -S hello_world.c
```

```c
/*
 * "Hello, World!": A classic.
 */

#include <stdio.h>

int
main(void)
{
        puts("Hello, World!");
        return 0;
}
```

```
    .section    __TEXT,__text,regular,pure_instructions
    .macosx_version_min 10, 10
    .globl  _main
    .align  4, 0x90
_main:                                  ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0, -4(%rbp)
    callq   _puts
    xorl    %ecx, %ecx
    movl    %eax, -8(%rbp)          ## 4-byte Spill
    movl    %ecx, %eax
    addq    $16, %rsp
    popq    %rbp
    retq
    .cfi_endproc

    .section    __TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
    .asciz  "Hello, World!"


.subsections_via_symbols
```

PROGRAM CONTROL STRUCTURES

- Sequence, Selection, Iteration
- An algorithm can be developed to solve any problem by using only these 3 program control structures
- You will find these structures implemented in many languages
- Being able to identify them across languages significantly simplifies the task of learning new languages

PROGRAM CONTROL STRUCTURES

- SEQUENCE

```
STATEMENT A
STATEMENT B
STATEMENT C
```

Sequence Control Structure

Major Tasks: Do Payroll

DO Enter Pay Details
DO Calculate Pay
DO Print Pay Details
DO Update Employee Records

Subtask: Enter Pay Details

INPUT employee name
INPUT weekly hours
INPUT rate of pay

Subtask: Calculate Pay

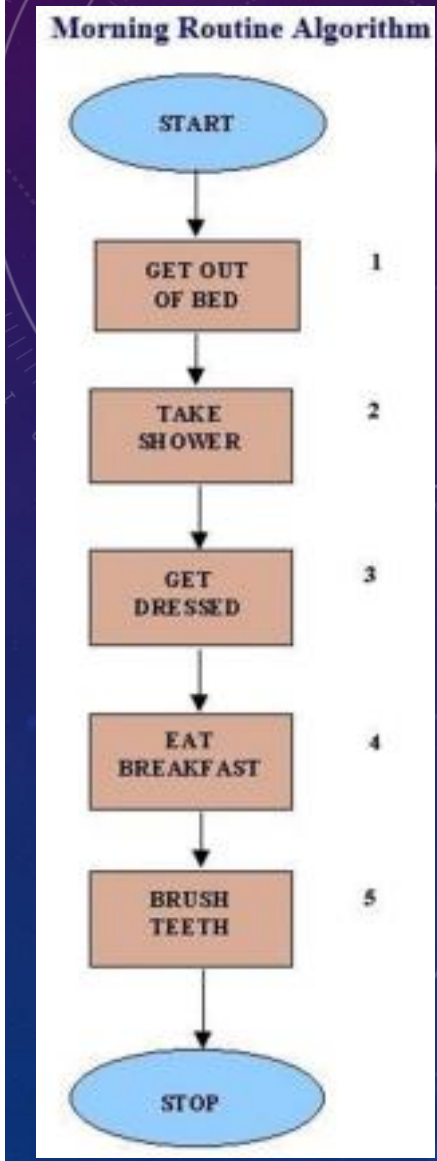
CALCULATE gross = hours * rate
CALCULATE tax = gross * .23
CALCULATE net = gross - tax

Subtask: Print Pay Details

PRINT employee name
PRINT gross
PRINT tax
PRINT net

Subtask: Update Employee Record

OPEN employee record
WRITE employee pay details
UPDATE year to date figures



PROGRAM CONTROL STRUCTURES

- SELECTION

```
IF Condition is true THEN
  DO Process A
ELSE
  DO Process B
ENDIF
```

Take a Shower

```
WASH Self
IF your hair is dirty THEN
    WASH your hair
ENDIF
DRY Self
IF you washed your hair THEN
    DRY your hair
ENDIF
APPLY deodorant
```

PROGRAM CONTROL STRUCTURES

- ITERATION

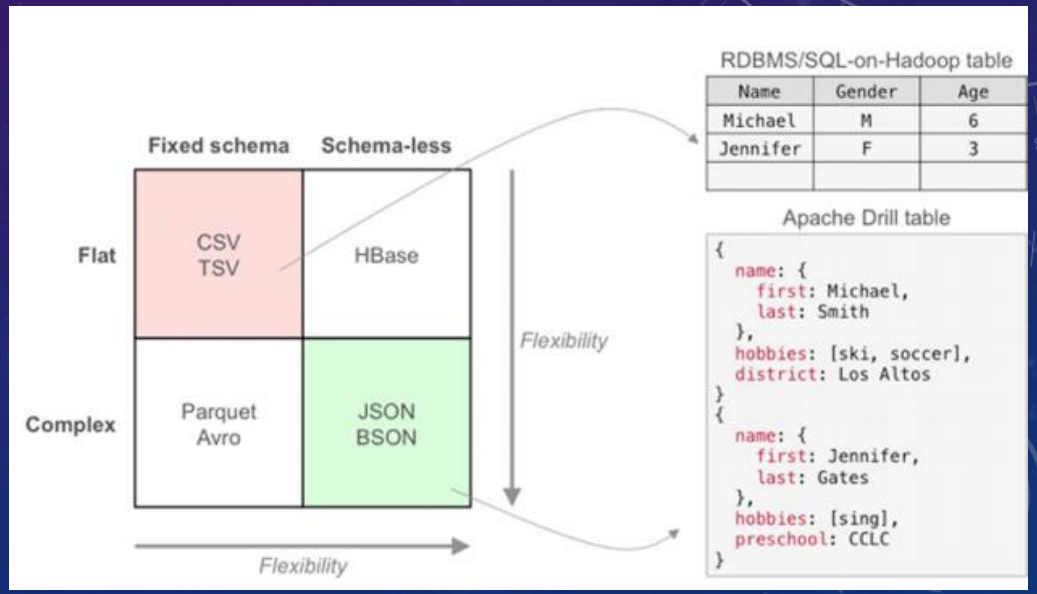
```
WHILE Condition is true DO
    Process statements
ENDWHILE
```

Do Payroll

```
WHILE there are more employees to process
DO
    ENTER pay details
    CALCULATE pay
    PRINT pay details
    UPDATE employee record
    DISPLAY 'Do more?'
ENDWHILE
```

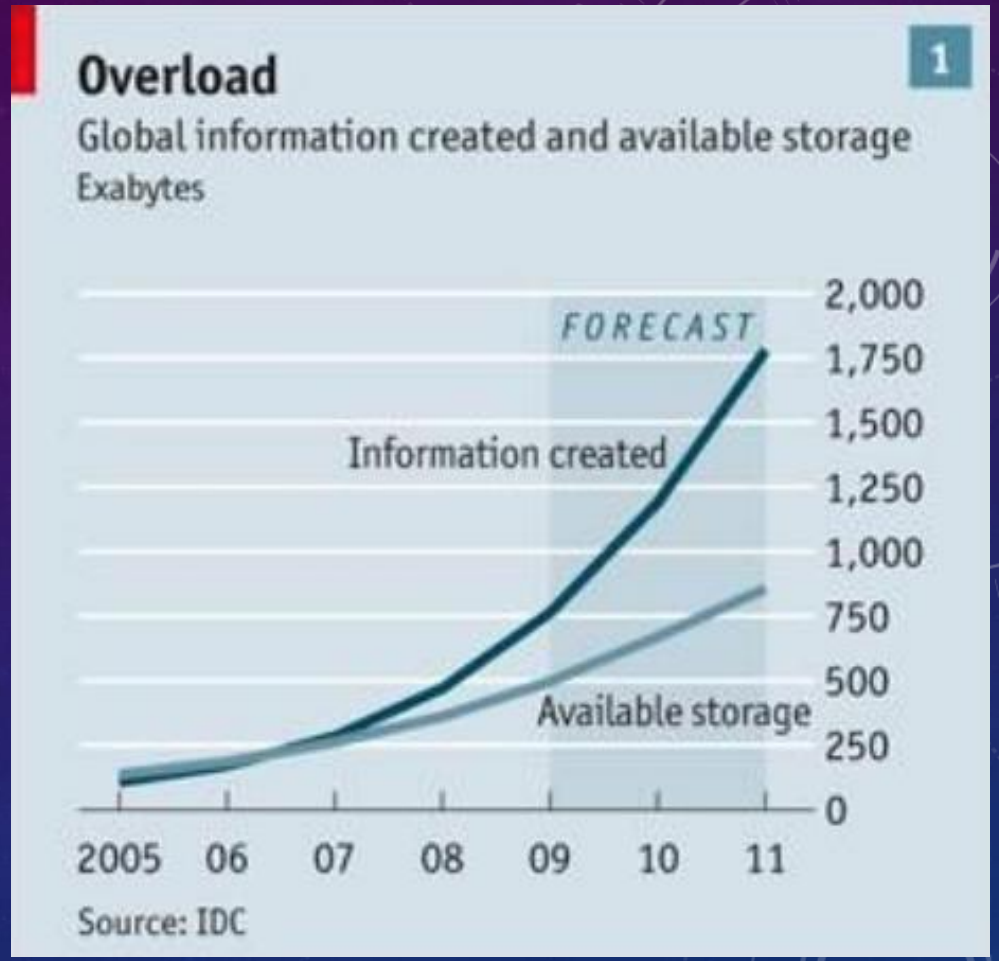
DATA

- Structured
  - Data organized into discrete fields, often with defined types, also known as “fixed schema”
    - Spreadsheets, databases
    - Classic analysis – Bayesian or generalized linear regression
  - Unstructured
    - Data that is not organized in any fashion
      - Free text, images, audio files, video
      - Natural Language Processing, image recognition, speech recognition
  - Semi-structured
    - Data that has some organization but is fluid rather than static, also called “self-describing” or “schema-less”
      - XML, JSON



SOURCES OF DATA

- http://www.data.gov/
- http://data.worldbank.org/
- http://data.un.org/
- http://mlr.cs.umass.edu/ml/index.html
- http://www.kdnuggets.com/datasets/index.html
- http://www.statsci.org/datasets.html
- And many, many more



This figure is from a 2010 article in The Economist. Retrieved online from: http://www.economist.com/node/15557443 May 9, 2017