

COMP301

Programming Languages Concepts

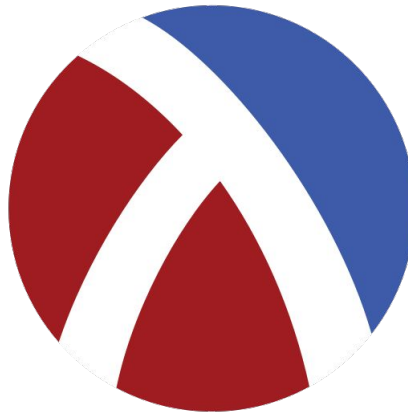
Assoc. Prof. T. Metin Sezgin

Outline

- Introduction
 - Course Description
 - Course Objectives
 - Learning Outcomes
- Syllabus
- Teaching Assistants
- Grading
 - Quiz
 - Project
 - Participation
- Problem Sessions

Introduction / Course Description

Programming languages (i.e. C++, Java, Ada, Lisp, ML, Prolog) concepts and paradigms. Syntax, semantics. Abstraction, encapsulation, type systems, binding, run-time storage, sequencers, concurrency, control. Providing examples from functional, object-oriented and logic programming paradigms.



Fun quiz activity – Please login



Fun poll activity

What kinds of programming paradigms are you familiar with?



Fun poll activity

What kinds of programming paradigms are you familiar with?



Fun quiz activity #1

What output does the following generate?



```
def create_adder(x):  
    tic = x  
  
    def adder():  
        tic = tic + 1  
        return tic  
  
    return adder  
  
fun_a = create_adder(0)  
fun_b = create_adder(0)  
  
print(fun_a(), fun_b(), fun_a(), fun_b())
```

Fun quiz activity #2

What output does the following generate?



```
def create_adder(x):  
    tic = x  
  
    def adder():  
        nonlocal tic  
        tic = tic + 1  
        return tic  
  
    return adder  
  
fun_a = create_adder(0)  
fun_b = create_adder(0)  
  
print(fun_a(), fun_b(), fun_a(), fun_b())
```


Fun quiz activity #3

What output does the following generate?



```
def create_adder(x):  
    global tic  
    tic = x  
  
    def adder():  
        nonlocal tic  
        tic = tic + 1  
        return tic  
  
    return adder  
  
fun_a = create_adder(0)  
fun_b = create_adder(0)  
  
print(fun_a(), fun_b(), fun_a(), fun_b())
```

Fun quiz activity #4

What output does the following generate?



```
def create_adder(x):  
    global tic  
    tic = x  
  
    def adder():  
        global tic  
        tic = tic + 1  
        return tic  
  
    return adder  
  
fun_a = create_adder(0)  
fun_b = create_adder(0)  
  
print(fun_a(), fun_b(), fun_a(), fun_b())
```

Introduction / Course Objectives

Teaching core programming concepts including data representation, procedural representation, grammars, environment models, parsing, evaluation, parameter passing, continuation passing, functional programming.

Introduction / Learning Outcomes

Students taking this class will gain fluency in core programming concepts including data representation, procedural representation, grammars, environment models, parsing, evaluation, parameter passing, continuation passing.

Syllabus

1. Functional programming

- a. Recursive programming
- b. Anonymous functions
- c. Constructors, accessors
- d. Scheme
- e. Data structures

2. Inductive Sets of Data

- a. Recursively Specified Data
- b. Deriving Recursive Programs
- c. Auxiliary Procedures and Context Arguments

3. Data Abstraction

- a. Specifying Data via Interface
- b. Representation Strategies for Data Types
- c. Interface for Recursive Data Types
- d. A Tool for Defining Recursive Data Types

4. Expressions

- a. Specification and Implementation Strategy
- b. LET: A Simple Language
- c. PROC: A Language with Procedures
- d. LETREC: A Language with Recursive Procedures
- e. Scoping and Binding of Variables
- f. Eliminating Variable Names
- g. Implementing Lexical Addressing

5. Continuation-Passing Interpreters

- a. A Continuation Passing Interpreter
- b. A Trampolined Interpreter

Syllabus

These subjects are from the books:

“Structure and Interpretation of Computer Programs” by Gerald Jay Sussman, Hal Abelson


“Essentials of Programming Languages” 3rd. ed. by Friedman and Wand.

Please find the books at:

<http://www.eopl3.com/> and

<http://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book.html>

What is Language?

lan·guage  *noun* \ˈlan-ɡwɪj, -wɪj\

Definition of LANGUAGE

- 1 **a** : the words, their pronunciation, and the methods of combining them used and understood by a community

b (1) : audible, articulate, meaningful sound as produced by the action of the vocal organs (2) : a systematic means of communicating ideas or feelings by the use of conventionalized signs, sounds, gestures, or marks having understood meanings (3) : the suggestion by objects, actions, or conditions of associated ideas or feelings
<language in their very gesture — Shakespeare> (4) : the means by which animals communicate (5) : a formal system of signs and symbols (as FORTRAN or a calculus in logic) including rules for the formation and transformation of admissible expressions (6) : MACHINE LANGUAGE 1
- 2 **a** : form or manner of verbal expression; *specifically* : STYLE

b : the vocabulary and phraseology belonging to an art or a department of knowledge

c : PROFANITY
- 3 : the study of language especially as a school subject
- 4 : specific words especially in a law or regulation



Why study programming languages?

- This class is not about
 - A particular programming language
 - Programming
 - Software engineering
 - Writing better code
- This class is about
 - Programming languages concepts
 - Data representation, recursion, binding
 - Control, state, parameter passing, scoping
 - Interpreters, compilers
 - Themes that span many languages
 - Some practical, some not so practical
 - Defining new languages
 - Learning new languages faster!
- This class is a Meta-Class

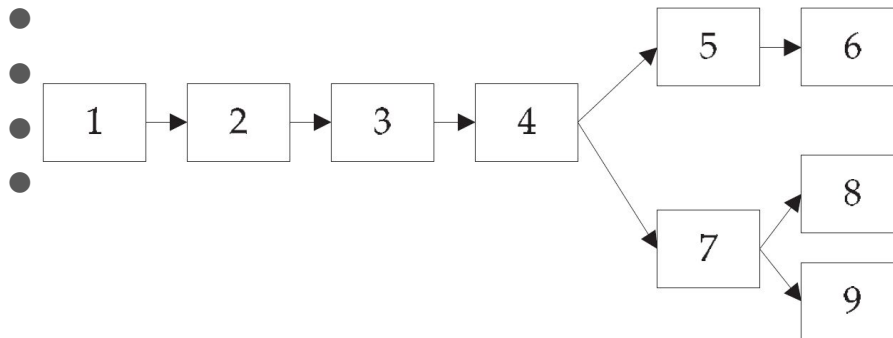


Organization of the course (textbook)



KOÇ
UNIVERSITY

- SICP Chapter 1: Building abstractions with procedures
- SICP Chapter 2: Building abstractions with data
- EOPL Chapter 1: Recursive data & recursive programming
- EOPL Chapter 2: Data types, abstraction
- EOPL Chapter 3: Interpreters
- EOPL Chapter 4: State, parameter passing
- EOPL Chapter 5: Control, trampolining, exceptions



COMP301

Details of the Syllabus

Important changes to the class

- Complete revamped
- Delivery
 - Revised for online delivery
 - The concept of “nuggets” or “pills”
 - Tight connection to a set of learning outcomes

Important changes to the class

- * Understand Functional Programming
- * Specify sets and data using bottom-up, top-down, rules of inference
- * Understand the differences between these specification
- * Understand and use BNF notation
- * Create recursive programs that operate on recursive data
- * Understand the basic principles for implementing procedures operating on recursively defined data
- * Specify data using define-datatype or equivalent
- * Manipulate data using the cases statement or equivalent
- * Understand data-structure and procedural representation of data
- * Understand concepts such as constructors, extractors, observers predicates
- * Understand and use formal notation for representation
- * Understand abstract and concrete syntax
- * Draw AST for given expressions
- * Understand the concept of behavior specification
- * Understand the difference between implementation and interface definition
- * Understand the concepts of expressed, and denoted values
- * Understand behavior specification for the LET, PROC, LETREC and EREF languages
- * Understand the pipeline that leads from lexical analysis to machine code
- * Understand the basic interpreter recipe and use it to build interpreters
- * Write interpreters for these languages
- * Understand the implementation of procedures and procedure application
- * Understand how recursive procedures can be implemented in an interpreter
- * Extend these languages with new expressions and features
- * Be able to specify the behavior of new expressions using formal notation
- * Implement new features
- * Understand the concept of scope, and variants such as dynamic and static scoping

- * Know the concepts of declaration and reference
- * Be able to write a simple translator
- * Be able to draw contour diagrams and match references to declarations
- * Understand lexical addressing, and find lexical depths
- * Be able to perform lexical conversion (lexical addressing)
- * Understand the concept of state and how a state-less language differs from one with effects
- * Understand the purpose of the environment and the store
- * Understand the implementation of store for implicit and explicit references

MIDTERM

- * Understand how pairs can be implemented, and do so
- * Explain why the second implementation is more efficient
- * Implement more sophisticated data structures (e.g., stack, arrays).
- * Understand that there are variations to parameter passing
- * Understand CBV/CBR and how they work
- * Understand the uses of CBR
- * Trace and CBV/CBR evaluation using the env & store
- * Implement CBR/CBR
- * Understand the philosophy of lazy evaluation
- * Understand call by need and call by name and how they work
- * Understand the uses of lazy evaluation
- * Trace and CBNeed CBRName evaluation using the env & store
- * Implement CBNeed CBRName
- * Understand the difference between tail-recursion & recursion
- * Understand the concept of continuation
- * Implement simple procedures using continuations (e.g., pow, fact, fib)
- * Understand how the concept of continuations can be applied to implement an interpreter
- * Implement an interpreter using continuations

FINAL

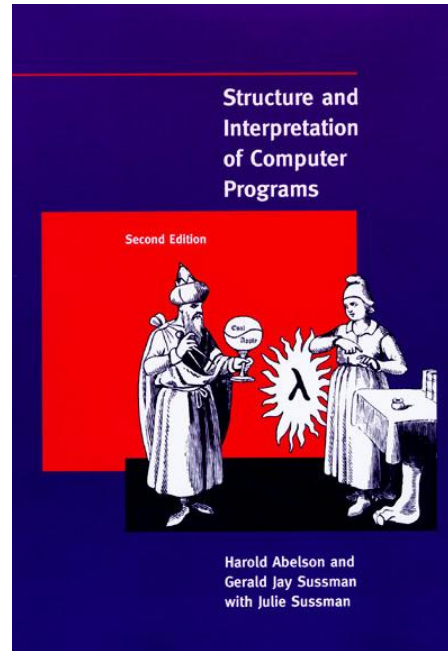
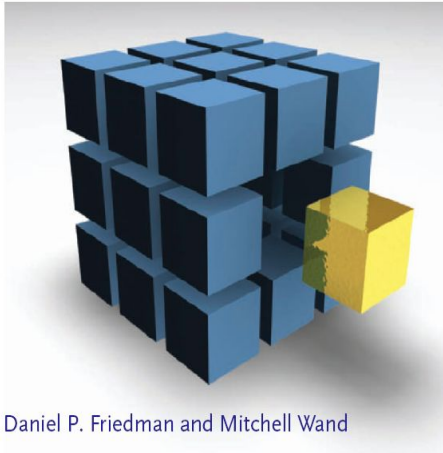
Important changes to the class

- Complete revamped
- Delivery
 - Revised for online delivery
 - The concept of “nuggets” or “pills”
 - Tight connection to a set of learning outcomes
- TA support
 - Extremely happy to have strong TA support
 - All class activities revised
- Assessment exercises
 - Completely revised
 - Project component added
 - Group work added

The Textbook

ESSENTIALS OF PROGRAMMING LANGUAGES

THIRD EDITION



Errata: <http://www.eopl3.com/errata.html>

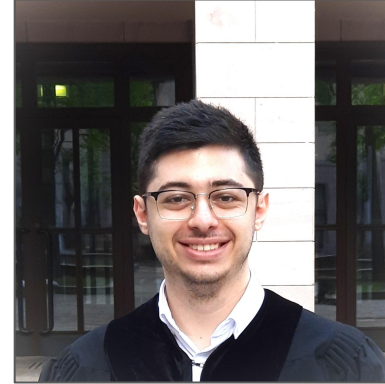
Teaching Assistants



Barış Batuhan Topal
baristopal20@ku.edu.tr

Office Hour: TBD

- Computer Science M.Sc. Student at Koç University
- Plays Bağlama, Likes Watching Movies & Anime.
- Working on Detection and Recognition Algorithms in Drawings.



Nazir Nayal
nnayal17@ku.edu.tr

Office Hour: TBD

- Computer Science M.Sc. Student at Koç University
- Interested in Chess, Anime and Competitive Programming
- Working on Computer Vision, Few-Shot Learning and Rare Events Detection in Autonomous Driving

Teaching Assistants

Ömer Faruk Tal

otal19@ku.edu.tr

Office Hour: TBD

- Computer Science Student at Koç University
- Interested in Chess, Sports, Drawing and Cartoons
- Interested in Deep Learning



Onur Eren Arpacı

oarpaci18@ku.edu.tr

Office Hour: TBD

- Computer Science Student at Koç University
- Interested in Drawing, US politics, and Anime
- Working on BlockChains, Smart Contracts and Decentralized Random Number Generation.

Teaching Assistants



Yeşim Hızır
yhizir19@ku.edu.tr
Office Hour: TBD

- Computer Science Student at Koç University
- Likes Movies, Baking and Cats
- Looking to learn more on distributed systems



Gökhan Sarı
gsari18@ku.edu.tr
Office Hour: TBD

- Computer Science and Mechanical Engineering Student at Koç University
- Interested in Art, Piano and Computer games

Dr. Metin Sezgin, Assoc. Prof.

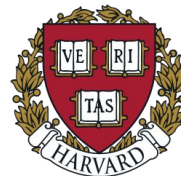
Director, Intelligent User Interfaces Group



MIT (MS '01) MIT (PhD '06)



postdoc



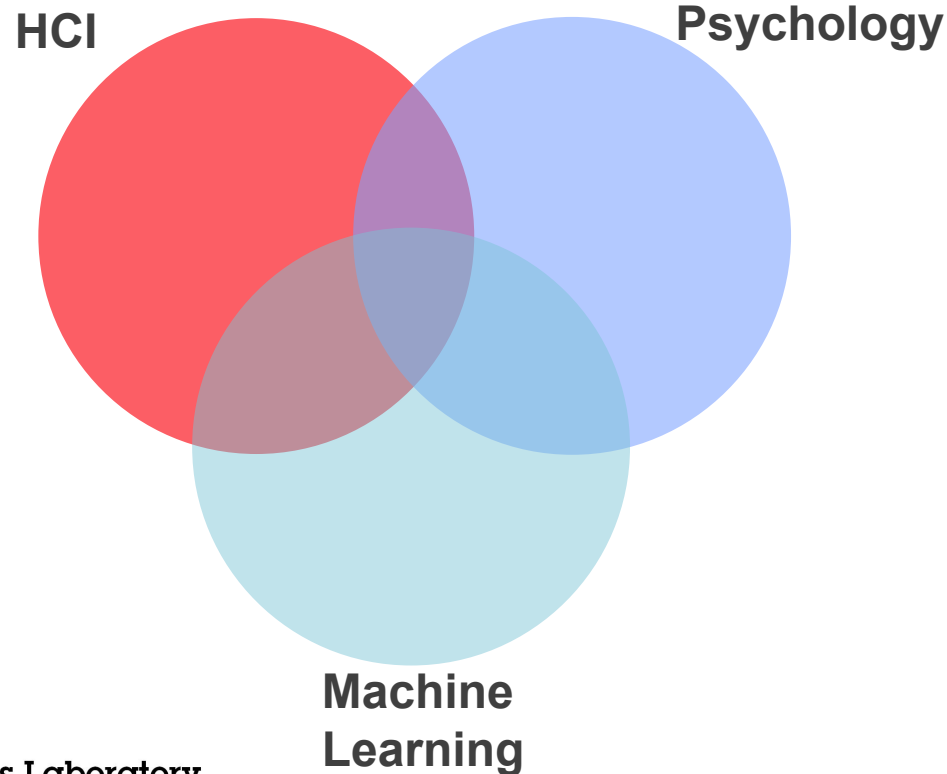
visiting appointments



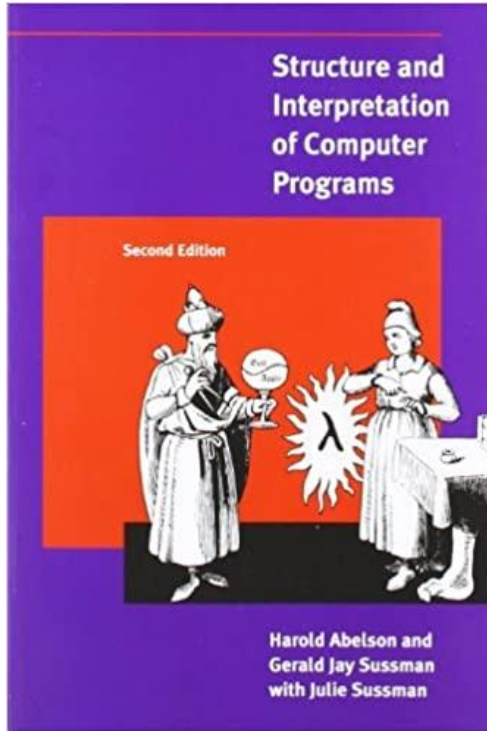
▪Areas of expertise

- Artificial Intelligence
- Intelligent User Interfaces
- Machine learning
- Multimodal interfaces

Intelligent User Interfaces

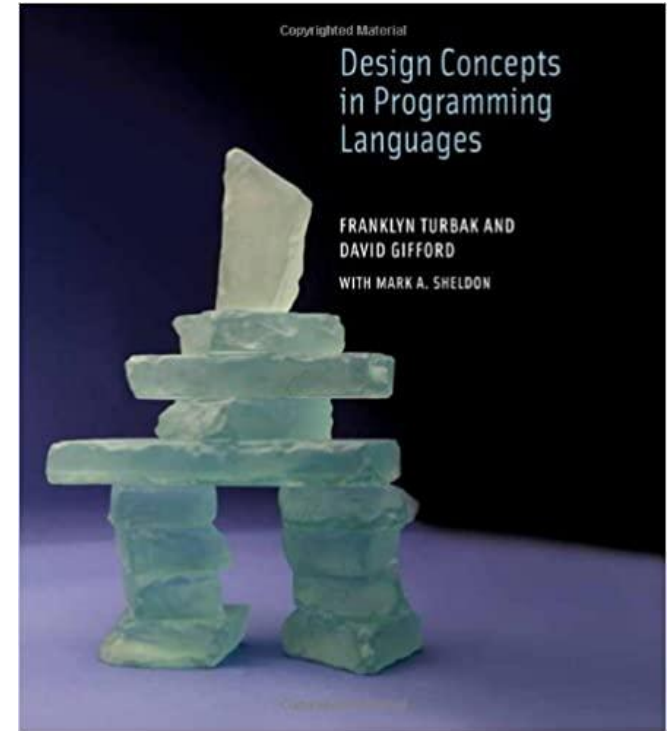
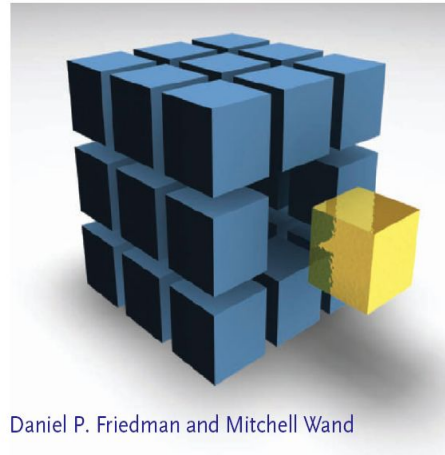


Why am I teaching this class?



ESSENTIALS OF PROGRAMMING LANGUAGES

THIRD EDITION



Office Hours

- Instructor
 - By appointment
- TAs
 - Office hours are 1 hour long.
 - Each week a different TA will hold an office hour, at their respective times.
 - These hours will also be declared separately and will be found in the Syllabus document.

Grading

Method	Description	Weight %
Quiz	10 quizzes	10,00
Project	Project	20,00
Midterm Exam	Midterm	25,00
Final Exam	Final	30,00
Participation	Participation	5,00
Laboratory	PS Sessions	10,00
Total:		100,00

Grading / Quiz

- There can be more than 10 quizzes, however, only the **top 10 highest score** will be counted.
- Quizzes will typically be made at the end of a lecture, not lasting more than 15-20 minutes in general.

Grading / Project

- Students will have a **1-2 week** time for each project.
- Submissions will include the **code**, a **report**, and a **self-assessment** within that report.
- Students can form a **group of 2 or 3**:
 - Those who **could not form a group** will be **randomly assigned one** by the TAs.
 - Students can **change/reform groups** for **each project** if they want to.
 - Students should speak to the TAs in case of a **bad-teammate**.
- **Projects are very important!** **Midterm** and **Final** questions will be **closely related** to the assets learned during the projects.

Grading / Participation

- Asking questions during the class, interacting with the class, answering questions etc. count greatly towards participation.
- Problem Sessions (PS) attendance counts towards participation. *More detail on the next slide.*

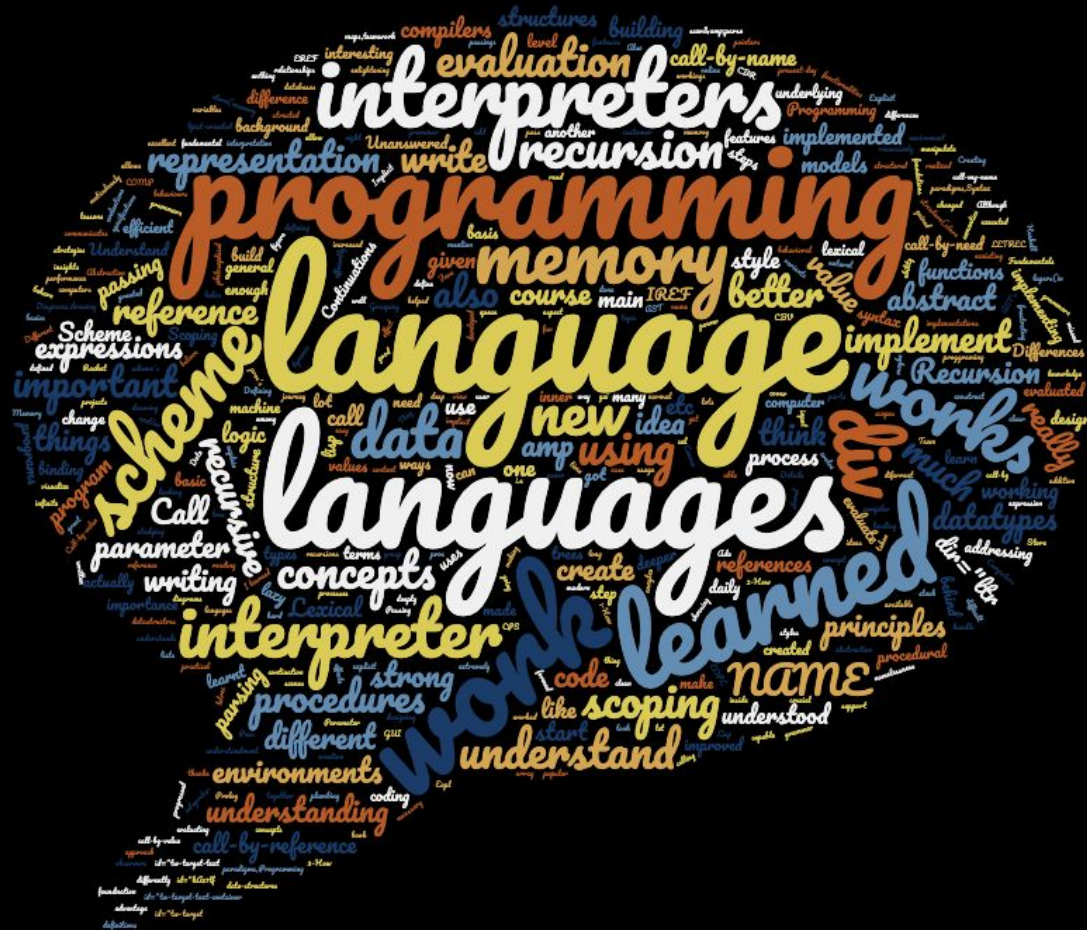
Problem Sessions

- Attendance to the PS'es are **mandatory**. They will contribute towards your participation.
 - You can miss **at most 2 PS'es** only.
- TA's will conduct the PS'es.
- It will be in a kind of Q&A format, where you can ask questions. We may also give you questions to solve during that PS.

Academic Dishonesty & Recording Disclaimer

See the University policy, declaration and the recording disclaimer.

Results of the Exit Survey



List three things you have learned in this course?



Give an example of an assignment or an activity that you think you did well to deserve the grade that you expect.





Good luck!

COMP301

Recursion

Lecture Overview



- **Recursively specified data**
 - Inductive data specification
 - Defining sets using grammars
 - Induction
- **Recursively specified programs**
 - Smaller sub-problem principle (wishful thinking)
 - Examples
 - Auxiliary procedures
- **Why?**
 - Procedures
 - Data

Recursion



- Recursion is important
 - Syntax in programming languages is nested
- Data definitions can be recursive
- Procedure definitions can be recursive

Recursion example



- Inductive specification of a subset of natural numbers $N = \{0, 1, 2, \dots\}$

Definition 1.1.1 *A natural number n is in S if and only if*

1. $n = 0$, or
2. $n - 3 \in S$.

- Which subset of N is this?
- Is 6 in S ?

Simple procedure for testing membership



- Write a procedure that follows the definition
- Remember the definition

Definition 1.1.1 *A natural number n is in S if and only if*

1. $n = 0$, or
2. $n - 3 \in S$.

- And the procedure

```
in-S? : N → Bool
usage: (in-S? n) = #t if n is in S, #f otherwise
(define in-S?
  (lambda (n)
    (if (zero? n) #t
        (if (>= (- n 3) 0)
            (in-S? (- n 3))
            #f))))
```

Simple procedure for testing membership



- More about the procedure

- Contract
- Domain
- Co-Domain (range)
- Usage
- Argument

```
in-S? :  $N \rightarrow Bool$   
usage: (in-S? n) = #t if n is in S, #f otherwise  
(define in-S?  
  (lambda (n)  
    (if (zero? n) #t  
        (if (>= (- n 3) 0)  
            (in-S? (- n 3))  
            #f))))
```