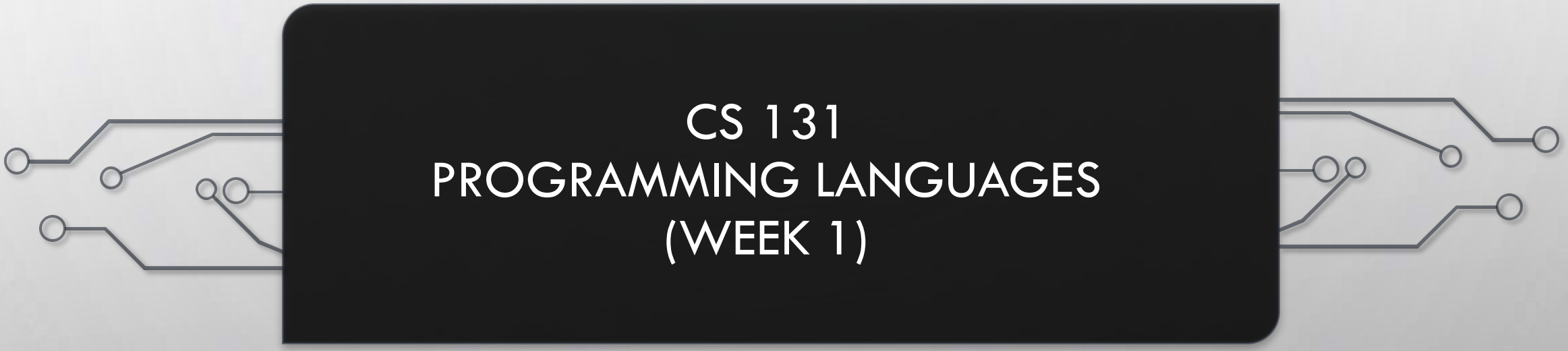


CS 131
PROGRAMMING LANGUAGES
(WEEK 1)

UCLA WINTER 2019

TA: SHRUTI SHARAN

DISCUSSION SECTION: 1D

ADMINISTRATIVE INTRODUCTION

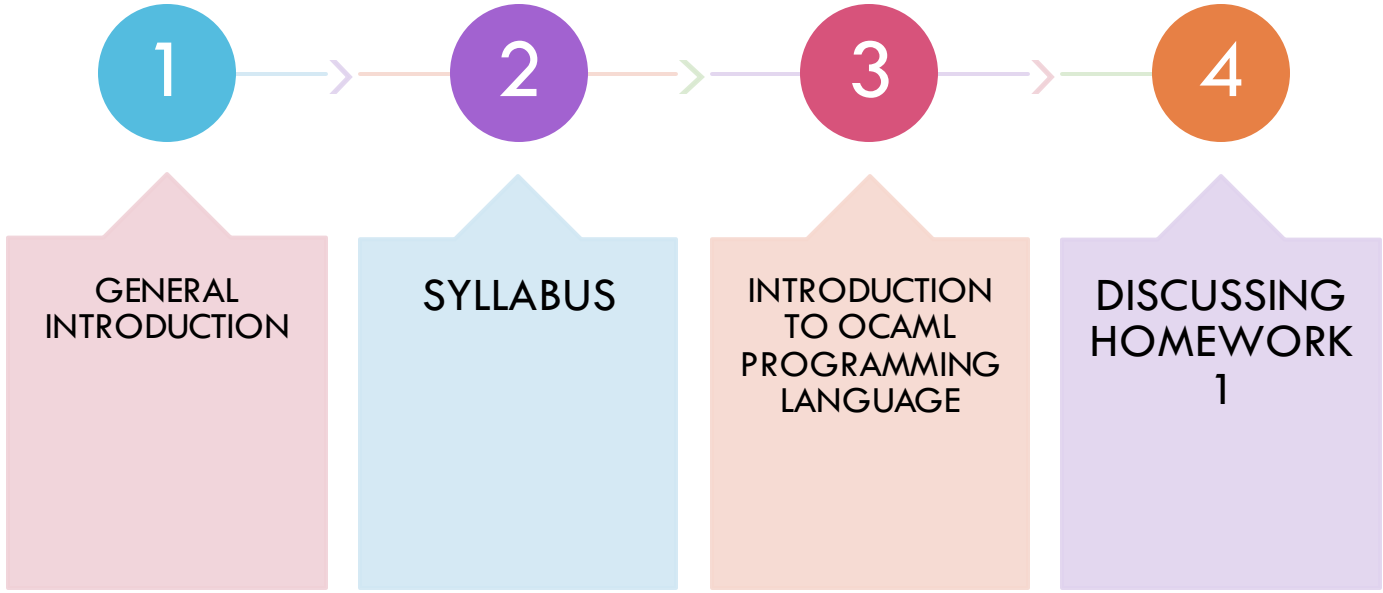
TA: Shruti Sharan

Email: shruti5596@g.ucla.edu
(Will reply by EOD)

Office Hours: Mondays 1.30PM – 3.30PM
Location: Eng. VI 3rd Floor

Discussion Section: Friday 4.00-5.50PM
Location: 2214 Public Affairs

TODAY'S AGENDA



RESOURCES

- Course Website: web.cs.ucla.edu/classes/winter19/cs131/
- Piazza: piazza.com/ucla/winter2019/cs131
- CCLE
- Course book:
  - Modern Programming Languages - A Practical Introduction
    - by Adam Brooks Webber

Working on homework yourself is important to success in CS131!

There is a high correlation between CS131 homework grades and the final class grade
Discussing general ideas is ok, sharing code or details is not.

Some homework will have automatic grading scripts

Not compiling -> No credit :(

Code must do exactly as the specifications say

Function signature must match your function signature, otherwise you will get no credit.

Make sure your code runs on the SeasNet servers before submission.

All homework related questions - ask on Piazza

If you are not sure, ask on Piazza
Possible to do anonymously, so that other students will not see your name
TAs/Professor will see your name though, to prevent spam.
Can use private note/question if not sure.

Homework will be submitted to CCLE

HOMEWORK

HOMEWORK

- First homework due on Wednesday (January 16th) 11:55pm!
- Second homework due Tuesday January 29th
  - Warning: Second homework will take significantly more time than the first homework.
    - Start working early, even though there is more time reserved for it.
- Other homework deadlines will be announced later. See course website .
- 6 Homework Assignments and a Project.
- Larger project at the end of the course
  - More details on this later

EXAMS

Midterm:

Thursday, 2018-02-07

during lecture

Final exam:

TBD

GRADING

- Homework 40%
  - Each homework has equal weight (5%)
  - project will be worth twice as much (10%)
  - Need a passing grade on homework to pass the course
  - Late homework will be penalized. Lateness penalty for an assignment that is submitted between *N* and *N*+1 full days late (where *N* is nonnegative) is 2^N % of the assignment's value.
- Midterm 20%
- Final exam 40%

DISCUSSION SECTIONS

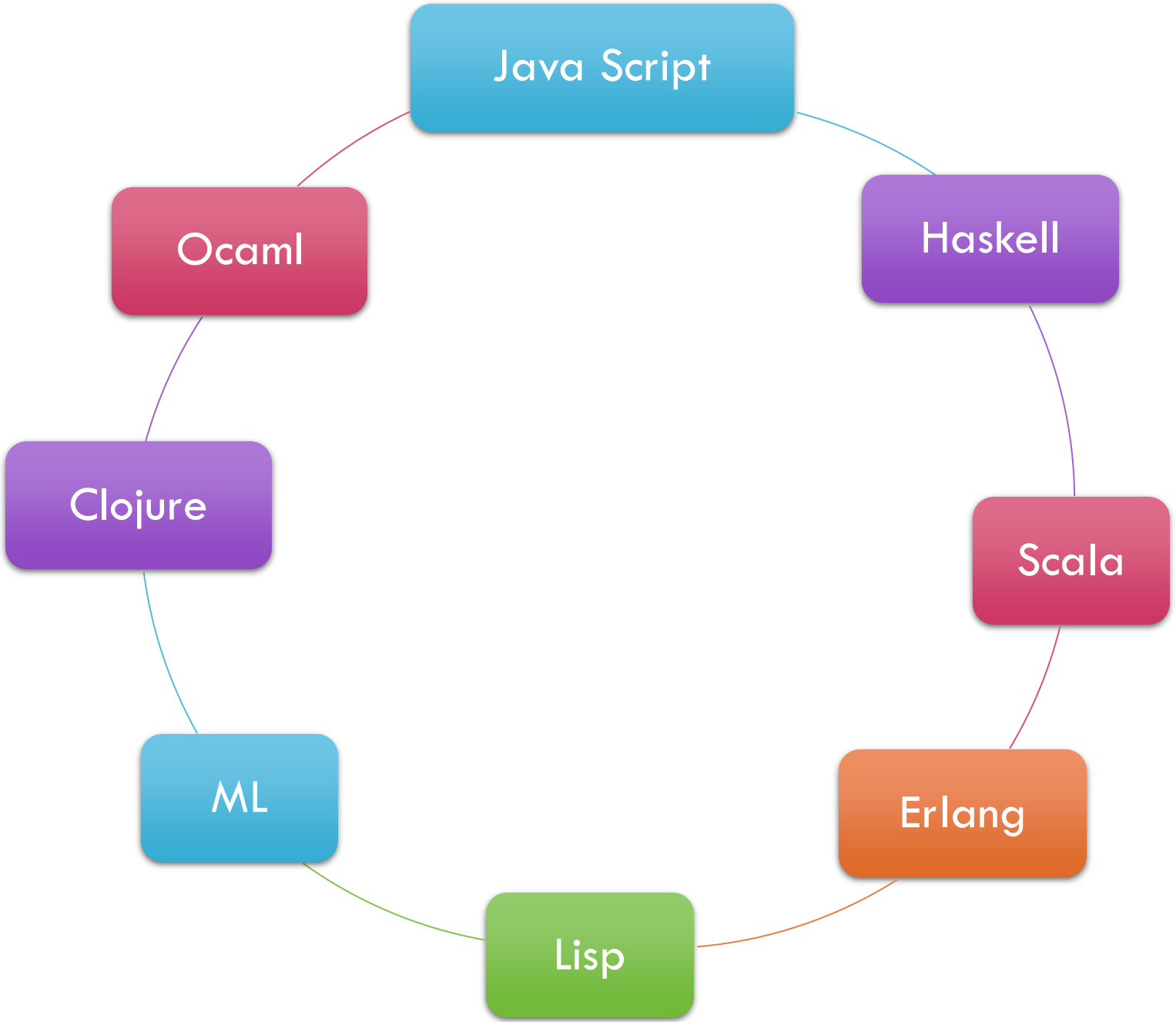
- Focus on teaching what is needed to solve the homework
  - E.g. Teaching the basics of programming languages that we use
- Course will use five languages: OCaml, Java, Prolog, Python, Scheme
  - New language approximately every two weeks!
  - Each language teaches different problem-solving techniques
  - First two homework will use OCaml
- If there is any Lecture Topic you want me to go over, please email before hand.
  - Use Office Hours for individual doubts.


QUESTIONS?

FUNCTIONAL PROGRAMMING LANGUAGE

- Programming paradigm in which we try to bind everything in pure mathematical functions style.
- Focus on describing what to solve, not how to solve it.
- Pure Functions
  - Immutable variables.
  - Function output will always be the same with the same input
- Recursion (No for/while loops) , no side effects
- First-Class and Higher order functions.

PROGRAMMING LANGUAGES THAT SUPPORT FUNCTIONAL PROGRAMMING LANGUAGE



OCAML PROGRAMMING LANGUAGE

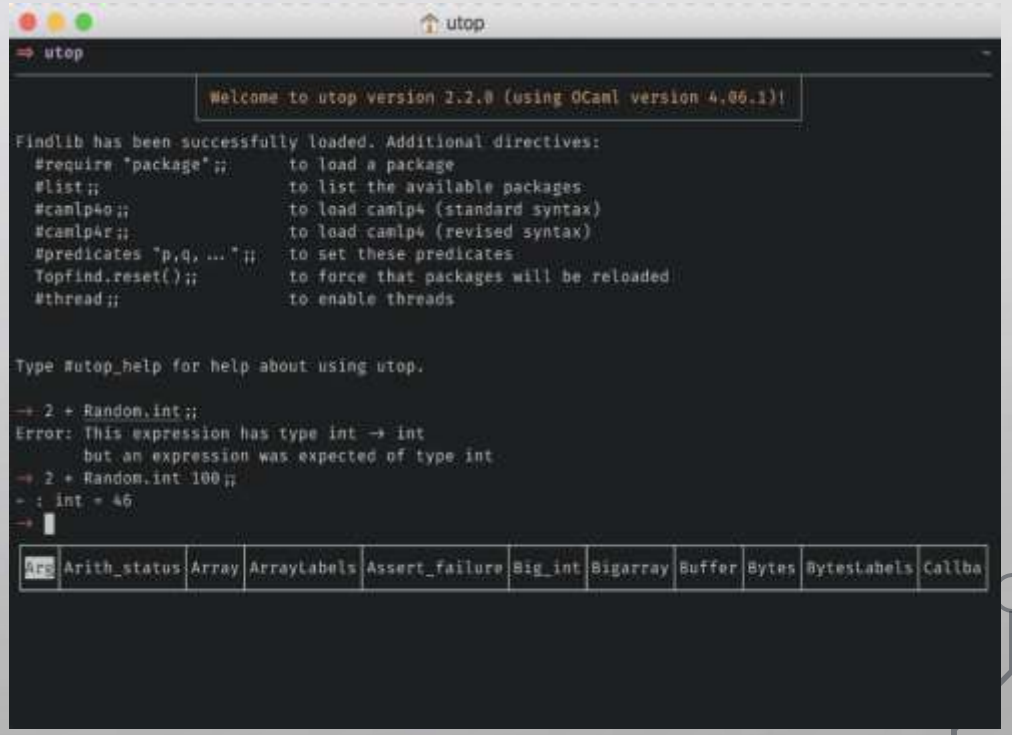
- Statically typed
  - Compiler/interpreter can warn you about many programming mistakes early on
  - Makes it faster to execute, as there is less need for safety checks when running the code
  - Easy to integrate with existing C/C++ libraries
- Small amount of code can achieve a lot!
  - Less code -> Less bugs

INSTALLING OCAML

- Installation instructions: https://ocaml.org/docs/install.html
  - Make sure you are using version 4.07.0
- You can use SEASnet servers too
  - lnxsrv06.seas.ucla.edu, lnxsrv07.seas.ucla.edu, lnxsrv09.seas.ucla.edu, and lnxsrv10.seas.ucla.edu
  - If you don't have a SEASnet account, apply for one ASAP: https://www.seas.ucla.edu/acctapp/
- Make sure that the OCaml version is correct (ocaml --version should show 4.07.0)
  - If not, check that /usr/local/cs/bin is in your path (which ocaml)
  - Instructions for this are at the course website under homework #1

ALTERNATIVE TOPLEVEL - utop

- utop is an improved toplevel (i.e., Read-Eval-Print Loop) for OCaml.
- https://opam.ocaml.org/blog/about-utop/
- https://github.com/ocaml-community/utop
- It can run in a terminal or in Emacs.
- It supports line editing, history, real-time and context sensitive completion, colors, and more.
- Not necessary, but makes coding easier.



FIRST PROGRAM

```
        OCaml version 4.07.1

# print_string "Hello World!\n";;
Hello World!
- : unit = ()
```

- print_string ~ is the function that is called
- "Hello World!\n" ~ parameter of the function
- Statement ends with two semi-colons (;;)
  - Necessary when using the interactive session, not needed in code files.
- Next line is printed by the function
  - Last line is the return value (unit), which conveys no information in this case

COMMENTS

```
# (* This is a comment *)
  (* This
* is
* a
* very
* long
* comment *)
  (* Nested (*comments*) are allowed too *)
```

- Please use comments in your code for documenting

BASIC DATATYPES

OCAML TYPE	RANGE
Int	31-bit signed int (roughly +/- 1 billion) on 32-bit processors, or 63-bit signed int on 64-bit processors
float	IEEE double-precision floating point equivalent to C's double
bool	A Boolean written either true or false
char	An 8-bit character
string	A string
unit	Written as ()

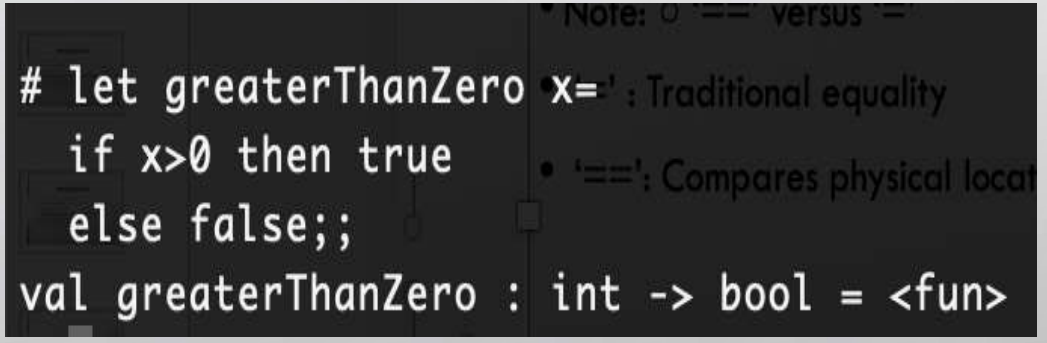
VARIABLES

- **Variables are immutable.**
- OCaml supports mutable variables too, but they should not be used in the homework.
- You can make them mutable by using the keyword **ref**.

```
# (*Global variable declaration *)
  let sevenEleven = 711 ;;
val sevenEleven : int = 711
# (* Local variable declaration *)
  let fortyTwo =
  let six = 6
  and seven = 7
  in six * seven ;;
val fortyTwo : int = 42
```

CONDITIONALS

- if-else are expressions.
- Note:
  - '==' versus '='
  - '=' : Traditional equality
  - '==': Compares physical location



FUNCTIONS

- Let <name> <parameter> = <expression> ;;

```
# let average a b =
    (a + b) / 2;;
val average : int -> int -> int = <fun>
```

- Functions can be parameters to other functions. (Higher order functions)
- Functions can be returned from a function.

LAMBDA FUNCTIONS

- Lambda functions (aka Anonymous functions) are not bound to any name
- Useful when using a function as a function parameter
  - Very common in functional programming!
  - “Higher-order function”

```
# (fun x -> x*x) 5;;
- : int = 25
```

RECURSION

- Must explicitly define using keyword **rec**
- Building block of truly functional solutions

```
# let rec factorial a =
    if a = 1 then 1 else a * factorial (a-1);;
val factorial : int -> int = <fun>
```

```
# factorial 5;;
- : int = 120
```

PATTERN MATCHING

- Syntax:

match expression

| p_1 -> expr_1

:

| p_n -> expr_n

```
# let one_many x =
  match x with
    0 -> "zero"
  | 1 -> "one"
  | _ -> "many" ;;
val one_many : int -> string = <fun>
# one_many 1 ;;
- : string = "one"
# one_many 100 ;;
- : string = "many"
```

```
# let rec bar i = match i with
  | 0 -> 0
  | 1 -> 1
  |
  j -> bar (j-2) + bar (j-1);;
val bar : int -> int = <fun>
```

PATTERN MATCHING

- More powerful version of the switch statement used in some other languages
- Pattern matching allows you to list all the different cases in a clean way
  - Underscore (_) matches any value that does not match the earlier rules
  - Cleaner than conditionals when there is a large number of possible cases
- Compiler lets you know which cases do not match to any of the rules
- Patterns can also include conditions using when keyword:

```
# let rec factorial a = match a with
    x when x < 2 -> 1
  | x -> x * factorial (x-1);;
```

- Evaluates a condition after the pattern match

USEFUL MODULES

- In the first homework, you are allowed to use two modules:
  - List
  - Pervasives
- Pervasives module provides the core functionality of OCaml
  - No need to explicitly import this module
- List module contains functions that are useful when operating with lists
  - This module is not imported by default!
- Before you can call a function that is in a module, you need to import it: open List
- Alternatively, you can add the module name into the function call, e.g. List.filter

PERVASIVE MODULE

- Exception handling
- Comparisons (=, >, <>, <=, >=)
- Boolean operators (!, &&, ||)
- Output functions (print_int, print_string, …)

For an exhaustive list, see

https://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html

LISTS

- Lists are defined using square brackets and the values are separated using semi-colons
- let numbers = [1; 2; 3; 4; 5]
- Under the hood, lists are immutable singly-linked lists
  - i.e. iterating them is fast, but random access is slow
- List consists of a head and a tail
  - Accessing these elements can be done with List.hd and List.tl -

```
# let numbers = [1;2;3];;
val numbers : int list = [1; 2; 3]
# 0:: numbers;;
- : int list = [0; 1; 2; 3]
# 0::1::2::[3];;
- : int list = [0; 1; 2; 3]
# 1::2;;
Error: This expression has type int but a
n expression was expected of type
         int list
```

LIST MODULES

- **List.rev** - reverses a list
- **List.flatten** or **List.concat** - Concatenate the list
- **List.map** - Applies a function to each element of the list
- **List.filter** - Applies a boolean function to each elements of the list. Returns only once that returns true
- **List.mapi** - Applies a function that takes 2 arguments, index and value.
- **List.hd** - Return 1st element of the list
- **List.tl** - Return eveything but 1st element of the list
- **List.forall** - Checks if each element of the list is valid for a boolean function
- **List.exists** - Checks if at least 1 element of the list is valid for a boolean function
- **List.mem** - Checks if an element is equal to something in the list.

For an exhaustive list, see: https://caml.inria.fr/pub/docs/manual-ocaml/libref/List.htmlAA

USER DEFINED TYPES

- We can create our own types!
- The **type** keyword is used to define
- syntax is this:

  type *name* = *type_expression*

```
# type color = Blue| Purple | Red;;
type color = Blue | Purple | Red
# Blue;;
- : color = Blue
```


CONTEXT FREE GRAMMAR

GRAMMARS

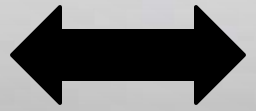
- **Symbol**
  - **Terminal:** A symbol which you cannot replace with other symbols
  - **Non-terminal:** A symbol which you can replace with other symbols
- **Rule**
  - From a non terminal symbol, derive a list of symbols
- **Grammar**
  - A starting symbol, and a set of rules that describe what symbols can be derived from a non terminal symbol

GRAMMARS

- **Symbols :** S, A, B, a, b
- **Terminal:** a, b
- **Non-terminal:** S, A, B
- **Starting symbol:** S

Rules

S -> A
S -> B
A -> aA
A -> a
B -> bB
B -> b



Rules

S -> A|B
A -> aA|a
B -> bB|b

How to Derive:

aaa

1. S
2. A (Apply rule S->A)
3. aA (Apply rule A->aA)
4. aaA (Apply rule: A->aA)
5. aaa (Apply rule: A->a)

UNREACHABLE RULES

Any rules that can **never be reached** from the start symbol by applying zero or more rules

Symbols : S, A, B, a, b, c
Terminal: a, b, c
Non-terminal: S, A, B, C
Starting symbol: S

Rules
S -> A|B
A ->A|aB|aA|a
B -> b
C -> c

What are the unreachable rule(s)?

BLIND ALLEY RULES

Any rule from which it is **impossible to derive a string of terminals.**

Symbols : S, A, B, a, b
Terminal: a, b,
Non-terminal: S, A, B
Starting symbol: S

Rules
S -> A|B
A ->A|aB|aA|a
B -> B

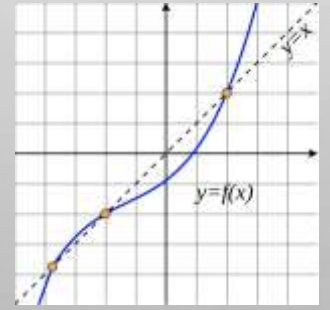
What are the blind alley rule(s)?

HOMEWORK #1 - HINTS

- Read and understand the given test cases
  - If the problem definition seems unclear, the test cases might help you understand how your code should behave
- Read through the documentation for List and Pervasives modules
  - Most problems can be solved with very little code if you don't reinvent the wheel
- Think whether the functions that you wrote for earlier problems can be used to solve the later problems
  - The power of functional programming comes from reusing very simple functions to implement more powerful functions

HOMEWORK #1 - HINTS

- For Q1 – Q5 use lists to represent sets and use all the functions from the Pervasives and List modules that were discussed.
- For Q6 you can assume that a fixed point (refer to diagram) can be found iteratively f(x), f(f(x)), f(f(f(x)))… so on.



  - some functions do not have a fixed point.
- Copy the type definition in your code file for Q7.

```
type ('nonterminal, 'terminal) symbol =
  | N of 'nonterminal
  | T of 'terminal
```

HOMEWORK #1 - SUBMISSION

- You are expected to submit 3 files:
  - hw1.ml - The functions that you implemented
  - hw1test.ml - Test cases for your functions
  - hw1.txt - Written assessment of how you ended up solving the problems the way you did - See course website for details

HELPFUL RESOURCES

- OCaml tutorial https://ocaml.org/learn/tutorials/
- Try OCaml https://try.ocamlpro.com
  - Interactive tutorial in your browser
  - Covers some topics that are not used in this course
- Useful Reading:
  - http://www2.lib.uchicago.edu/keith/ocaml-class/home.html
  - https://realworldocaml.org/
  - https://learnxinyminutes.com/docs/ocaml/
  - http://www.cs.princeton.edu/~dpw/courses/cos326-12/lec/02c-unit-option.pdf

QUESTIONS?

- Piazza - Fastest way to get answers
  - TAs, your professor, or classmates can answer your question, so this is the best channel to get help when you're stuck
- Come to office hours.