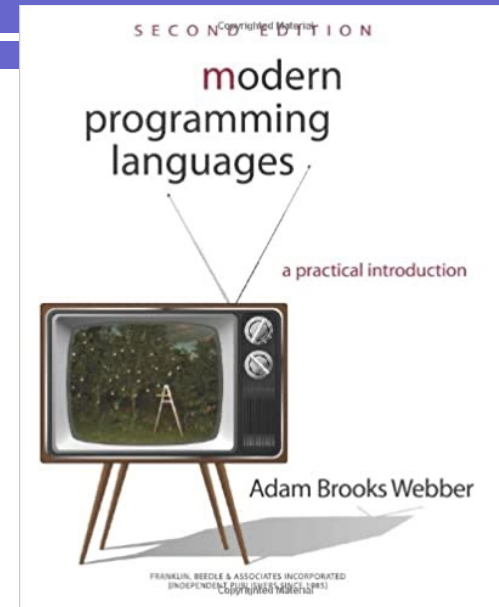
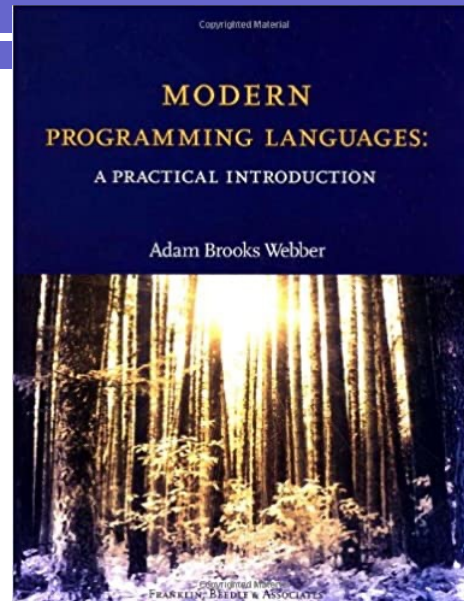
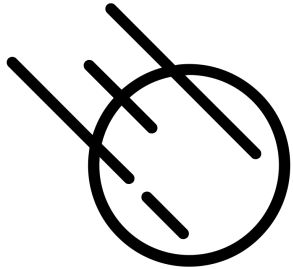


# Welcome - CSC 301



## CSC 301- Fundamentals of Programming Languages

- Instructor: Dr. Lutz Hamel
- Email: [lutzhamel@uri.edu](mailto:lutzhamel@uri.edu)
- Book: “Modern Programming Languages”, **any edition**
- (for more details see BrightSpace)

# Why Study Programming Languages?

- Amazing variety
  - ~2300 different programming languages discussed on online forums\*.
- “Strange” controversies
  - Should a programming language have a ‘goto’ statement?
  - Should an OO language support inheritance?
  - Terminology: argument vs. actual parameter.
- Many connections
  - Programming languages touch upon virtually all areas of computer science: from the mathematical theory of **formal languages** and **automata** to the implementation of **operating systems**.
- Intriguing evolution
  - Programming languages change!
    - New ideas and experiences trigger new languages.
    - New languages trigger new ideas, etc.

# Programming Language Classes

There are many different programming language classes, but three classes or paradigms stand out:

- Imperative Languages
- Functional Languages
- Logic/Rule Based Languages

# What Happened to OOP?

- Object-orientation is really a property of the type system of a language.
- OO features have traditionally been added to imperative languages (C++, Java, Python)
- Object-oriented features have also been added to:
  - Functional programming languages like Lisp (CLOS)
  - Logic languages like Prolog (Logtalk)
- Here we look at object-based programming within the multi-paradigm language Asteroid

# Meet Our Languages

- Asteroid – An object-based, imperative, and functional programming language being developed right here at URI
  - <https://asteroid-lang.org>
- Prolog – A logic programming language, most famously used in IBM Watson
  - The IBM Watson knowledge base was filled with 200 million pages of information, including the entire Wikipedia website. To parse the questions into a form that IBM Watson could understand, the IBM team used Prolog to parse natural-language questions into new facts that could be used in the IBM Watson pipeline. In 2011, the system competed in the game *Jeopardy!* and defeated former winners of the game.
  - <https://www.swi-prolog.com>

# Example Computation

- Recursive definition of the factorial operator

$$x! = \begin{cases} 1 & \text{if } x = 1, \\ x(x-1)! & \text{otherwise.} \end{cases}$$

for all  $x > 0$ .

# Imperative Languages

- Hallmarks: assignment and iteration
- Examples: C, FORTRAN, Imperative sublanguage of Asteroid
- Example Program: factorial program in (imperative) Asteroid

```
function fact with n do
  let val = 1.
  while n > 1 do
    let val = val*n.
    let n = n-1.
  end
  return val.
end
```

# Imperative Languages

## Observations:

- The program text determines the order of execution of the statements.
- We have the notion of a '*current value*' of a variable – accessible state of variable.

This is not always true in other languages.



# Imperative Asteroid

```
1  -- compute the factorial
2
3  load system io.
4  load system type.
5
6  function fact with n do
7    let val = 1.
8    while n > 1 do
9      let val = val*n.
10     let n = n-1.
11   end
12   return val.
13 end
14
15 let x = type @tinteger(io @input("Enter a positive integer: ")).
16 io @println ("The factorial of "+x+" is "+(fact x)).
```

# Functional Languages

- Hallmarks: recursion, multi-dispatch, single valued variables.
- Examples: ML, Lisp, Haskell, Functional sublanguage of Asteroid
- Example Program: factorial program in (functional) Asteroid

multi-dispatch {

```
function fact
  with 1 do
    return 1
  with n do
    return n*fact(n-1).
  end
```

} n is single valued variable.

recursion

# Functional Languages

## Observations:

- **No** explicit assignments necessary
  - we will allow them later for convenience sake but they will introduce only single valued variables
- The name stems from the fact that programs consist of *recursive* definitions of **functions**.

# Functional Asteroid

```
1  -- compute the factorial
2
3  load system io.
4  load system type.
5
6  function fact
7    with 1 do
8      return 1
9    with n do
10     return n*fact(n-1).
11  end
12
13  let x = type @tointeger(io @input("Enter a positive integer: ")).
14  io @println ("The factorial of "+x+" is "+(fact x)).
```

# Logic Programming Languages

- Hallmarks: programs consist of **rules** that specify the problem solution.
- Examples: Prolog, Maude, Isabelle
- Example Program: factorial program written in Prolog

rules {

```
fact(1,1).
fact(X,F) :-
    X1 is X-1,
    fact(X1,F1),
    F is X*F1.
```

fact(in,out)

'and'

# Logic Programming Languages

## Observations:

- Rules do *not* appear in the order of execution in the program text.
- No specific order of execution is given – rules ‘fire’ when necessary.

# Prolog

```
% factorial program
```

```
fact(1,1).
```

```
fact(X,F) :-
```

```
    X1 is X-1,
```

```
    fact(X1,F1),
```

```
    F is X*F1.
```

```
compute :-
```

```
    X is 3,
```

```
    fact(X,F),
```

```
    writeln(F).
```

# Object-Based Languages

- Hallmarks: bundle data with the allowed operations ➞ Objects
- Asteroid takes an interesting approach here – structures with functions.

```
1  -- simple object-based program
2
3  load system io.
4
5  -- define our rectangular structure with member functions
6  structure Rect with
7    data xdim.
8    data ydim.
9
10   -- return the area of the rectangle
11   function area with none do
12     return this @xdim * this @ydim.
13   end
14 end
15
16 let r = Rect(4,2).
17 io @println ("The area of rectangle <"+r @xdim+", "+r @ydim+"> is "+r @area()).
```



# Programming Language Classes

## General Observations:

- Programming languages guide programmers towards a particular programming style:
  - Imperative → iteration/assignment
  - Functional → mathematical functions
  - OO → objects
  - Logic → rules
- Programming itself guides the developer towards new language ideas:
  - Recursion was introduced by John McCarthy in the 1950' s with the programming language Lisp to solve problems in AI.
  - Classes and objects were developed by Nygaard and Dahl in the 1960' s and 70' s for the language Simula in order to solve problem in simulations.

# Take Away

- There exist many programming languages today (> 2000)
- In order to understand the similarities and differences  $\Rightarrow$  sort into *classes*
  - Imperative
    - assignment and iteration
  - Functional
    - Recursion, single valued variables
  - Logic/rule based
    - programs consist of rules
- Object-based
  - bundle data with the allowed operations

# Reading & Assignments

- Reading: Modern Programming Languages (MPL) Chap 1.
- Reading: Asteroid User Guide
  - <https://asteroid-lang.readthedocs.io>
- Assignment #0: Download & Read Syllabus – upload a copy of it into BS