AA 2014-2015

PROGRAMMAZIONE 2

1.Introduzione

PRESENTAZIONI

- Gianluigi Ferrari
  - Email giangi@di.unipi.it
  - Web www.di.unipi.it/~giangi
- Di cosa mi occupo (ricerca)
  - Formal methods in Software Engineering
    - ✓ Verification, model checking, and static analysis of programs
  - Programming languages & models for Concurrent and Distributed Systems
    - ✓ Service oriented computing
    - ✓ Cloud computing
  - Security
    - ✓ Language-based security

PRESENTAZIONI

- Fabio Gadducci
  - Email fabio@di.unipi.it
  - Web www.di.unipi.it/~gadducci
- Di cosa mi occupo (ricerca)
  - Formal methods in Software Engineering
    - ✓ Verification, model checking, and static analysis of programs
  - Programming languages & models for Concurrent and Distributed Systems
    - ✓ Service oriented computing
    - ✓ Theoretical foundations
  - Visual modeling
    - ✓ Graphical specifications and model transformations

UNIVERSITÀ DI PISA

PROGRAMMAZIONE 2

Cosa studiamo?

Due tematiche principali

Programmazione OO

- Tecniche per la programmazione orientata ad oggetti (in piccolo)
  - Specifica, implementazione, correttezza
    - ✓Dimostrare la correttezza di una implementazione è tanto importante quanto programmare
  - Programmazione concorrente
- Esemplificate utilizzando Java
  - non è compito di questo corso introdurre il linguaggio nella sua interezza…
  - né tanto meno le sue librerie (che imparerete da soli, quando vi servono)

Una valanga di libri…

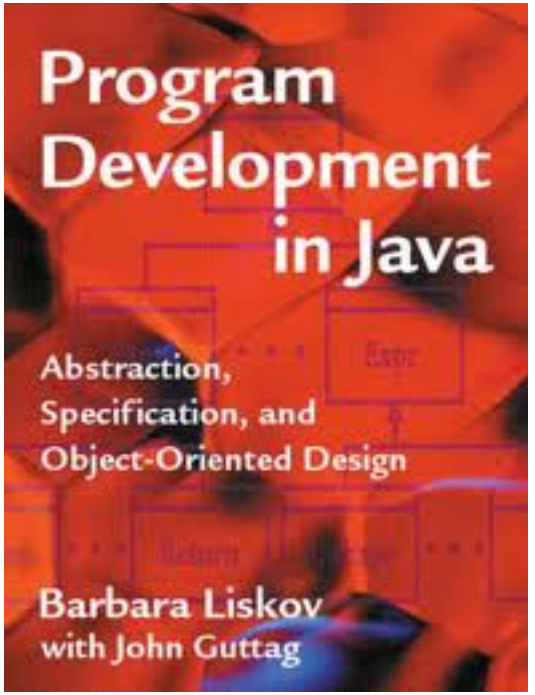
Materiale didattico

B. Liskov, J. Guttag

Program development in Java

(Addison Wesley 2000)

Datato, ma copre tutti gli aspetti fondamentali

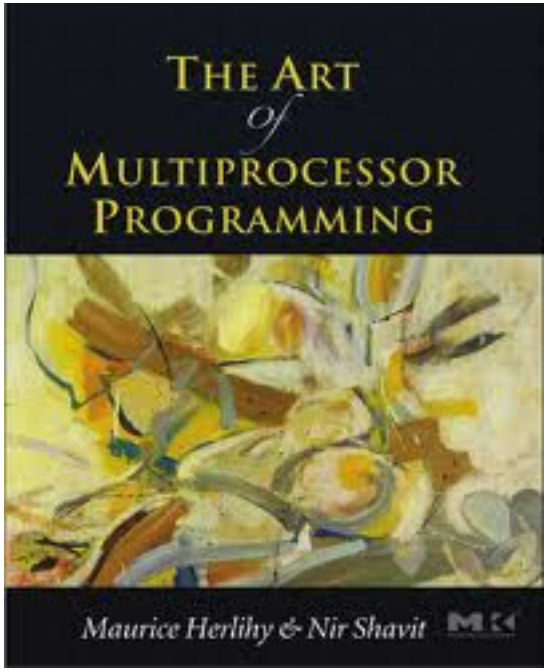


Materiale didattico

M. Herlihy, N. Shavit

The art of multiprocessor programming

(Morgan Kaufmann 2012)



Programmazione concorrente e tecniche per multi-core

Materiale didattico

R. Bruni, A. Corradini,
V. Gervasi

Programmazione in Java

(Apogeo 2011)

Ottima introduzione per
chi pensa di avere lacune
con la programmazione

Online

- *Oracle Java tutorials*, docs.oracle.com/javase/tutorial/java/
- **David Eck**, *Introduction to programming using Java*, math.hws.edu/javanotes/
- Online ne trovate molti altri…
- …sentitevi liberi di seguire la vostra curiosità

Il nostro obiettivi

- Nessun linguaggio è perfetto e prenderemo in esame una piccola parte di Java
- **Obiettivo 1**: acquisire competenze generali che possano essere applicate a una varietà di linguaggi di programmazione.
- **Obiettivo 2**: acquisire le competenze per imparare “presto e bene” un nuovo linguaggio di programmazione.

Linguaggi di Programmazione

- Studiare i principi che stanno alla base dei linguaggi di programmazione
- Essenziale per comprendere il progetto, la realizzazione e l'applicazione pratica dei linguaggi
- Non ci interessa rispondere alla domanda "Java è meglio di C#"?

Tanti aspetti importanti…

- **Paradigmi linguistici**
- **Semantica operazionale**
- **Implementazione**: strutture a tempo di esecuzione
- Il nostro approccio: la descrizione dell'implememementazione del linguaggio è guidata dalla semantica formale!
  - Stretta relazione tra la semantica e la struttura del run time del linguaggio
  - Struttura del run-time simulata in OCaml (a volte ritornano)
- Ci sono numerosi libri sull'argomento che sono utili per il nostro corso… ma metteremo a disposizione delle note.

FONDAMENTI: UN VALORE

- Evitare discussioni da osteria
- Evitare malfuzionamenti
- Numerosi esempi
  - Post sul blog ufficiale di Microsoft Azure:
    - ✓ *Alle 17:45 ora del Pacifico del 28 febbraio 2012 Microsoft ha rilevato un problema che affligeva i servizi Windows Azure in diverse regioni. Il problema è stato analizzato rapidamente e attribuito a un bug software. Sebbene le origini effettive siano oggetto di indagine, il problema sembra fosse causato da un calcolo del tempo errato nell'anno bisestile*.
- **La teoria aiuta il progetto e la realizzazione dei linguaggi**
- **Esempio**: implementazioni efficienti si possono ottenere se la generazione del codice eseguibile è ritardata fino a che non sono disponibili dati del run-time

Materiale didattico

M. Gabbrielli, S. Martini

Linguaggi di programmazione

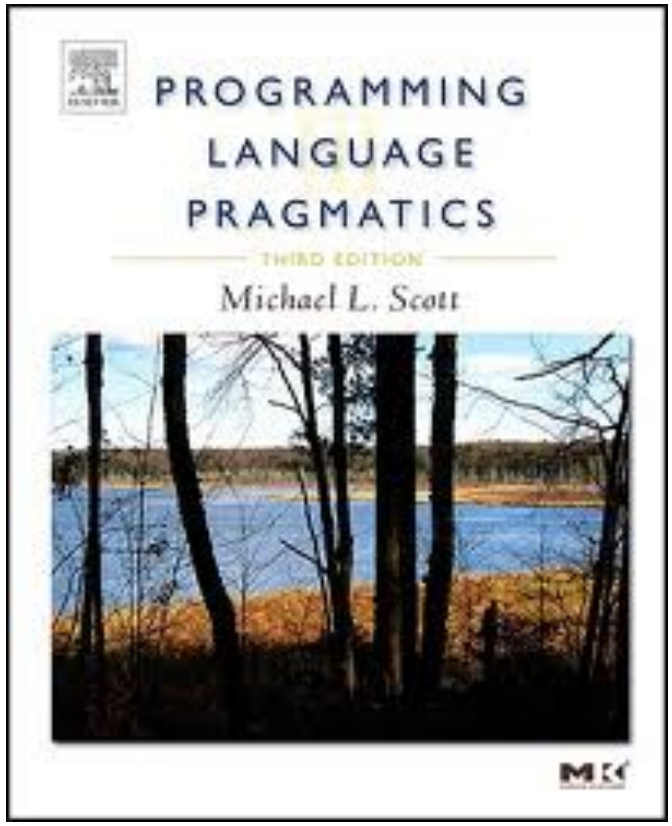
(McGraw-Hill 2006)

Materiale didattico

M. Scott

Programming language pragmatics

(Morgan Kaufmann 2009)

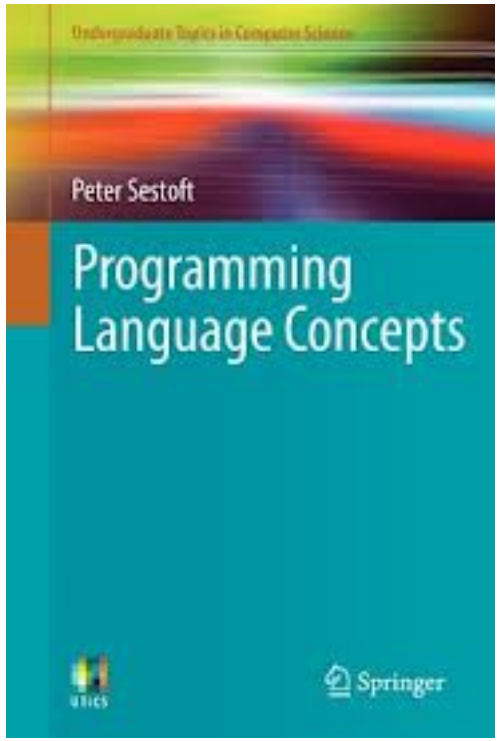


Materiale didattico

P. Sestoft

Programming language concepts

(Springer 2012)



PR2: istruzioni per l'uso

- Il materiale didattico delle lezioni sarà disponibile sulla pagina web così come tutti i programmi OCaml e Java che verranno discussi nelle esercitazioni
- **Prova di esame = progetto + prova scritta + orale**
  - ammissione all'orale con votazione >= 18/30 nello scritto & valutazione positiva del progetto.
  - le 2 prove intermedie possono sostituire la prova scritta
- **Consigli**
  - seguire il corso mantenendosi al passo con lo studio
  - partecipare (attivamente) a lezioni ed esercitazioni
  - sostenere le prove intermedie

Competenze richieste (nostre aspettative)

- Familiarità coi concetti base di programmazione funzionale (Caml) e imperativa (C)
  - Programmazione 1 e laboratorio & Logica per la programmazione
- Familiarità algoritmica e programmazione con le strutture dati di base (liste, pile, code, alberi, hash table, …)
  - Algoritmica e laboratorio

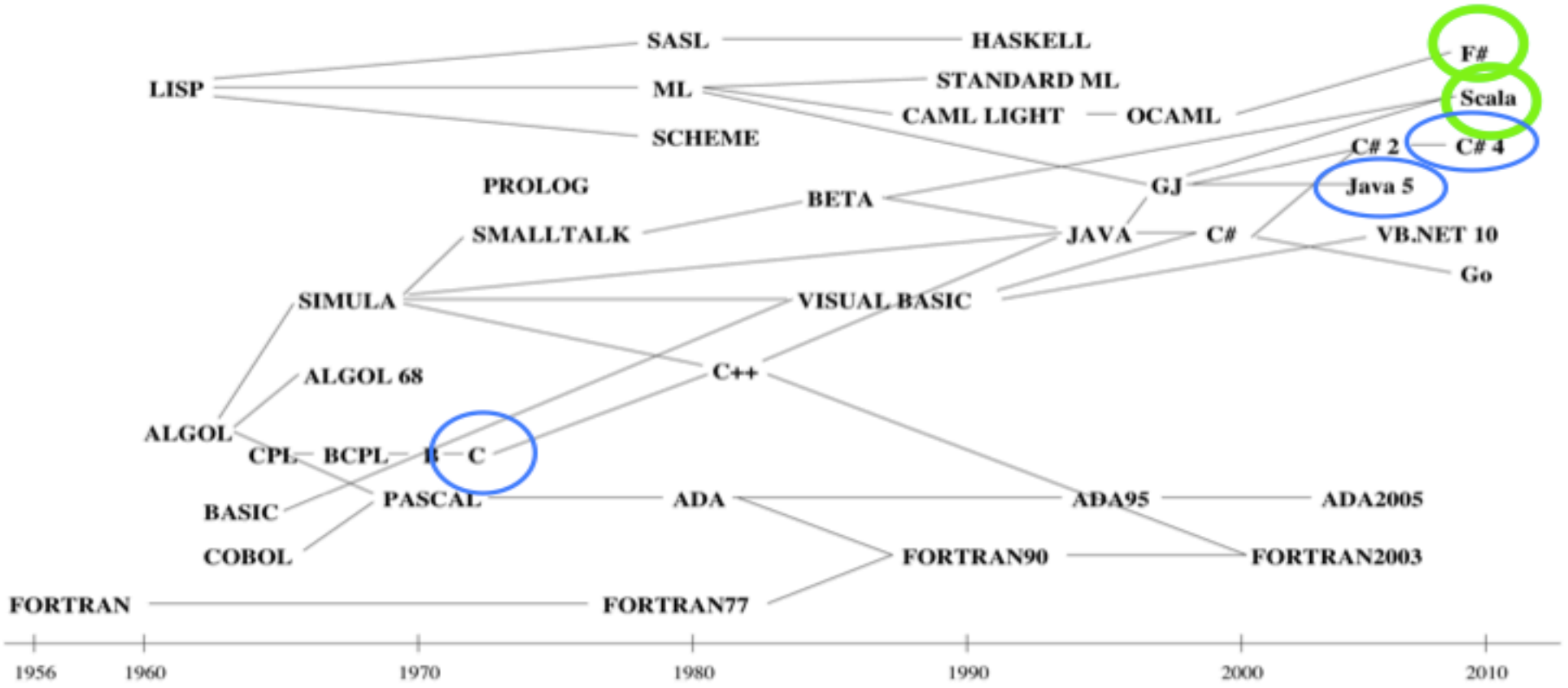
Linguaggi e astrazione

- I ***linguaggi di programmazione*** sono il più potente strumento di ***astrazione*** messo a disposizione dei programmatori

- I linguaggi si sono evoluti trasformando in costrutti linguistici (e realizzandoli una volta per tutte nell'implementazione)
  - **tecniche e metodologie** sviluppate nell'ambito della programmazione, degli algoritmi, dell'ingegneria del software e dei sistemi operativi
  - **settori di applicazioni** (basi di dati, intelligenza artificiale, simulazione, etc.)

- Di fondamentale importanza l'introduzione di ***meccanismi di astrazione***, che permettono di estendere un linguaggio ***programmando*** nuove operazioni, tipi di dato, etc.

Il diagramma evolutivo



Tanti linguaggi. Perché?

- Prendiamo il migliore e basta!!!
  - Come vedrete a **Calcolabilità e Complessità**, i linguaggi di programmazione sono tutti (Turing) equivalenti: stessa potenza espressiva
- I migliori sono tanti…
  - Visione Oracle-Sun: Java
  - Visione Microsoft: C#
  - Visione dello sviluppatore Web: JavaScript
- Tante motivazioni diverse: alcuni linguaggi meglio si adattano a un particolare contesto
  - PROLOG: AI

A day in the life of a web programmer

- Develop a web site
  - Separare presentazione, stile e funzionalità
- Client side programming
  - Javascript (funzionalità), HTML (presentazione), CSS (stile)
- Server side programming
  - CGI scripts
  - Scripting (PHP, Pearl, Ruby, …)
  - Java
  - Database access (SQL)
  - XML per web services

Navigate sul web

- Il sito

  www.scriptol.com/programming/fibonacci.php

  descrive il programma che calcola i numeri di fibonacci nei principali linguaggi di programmazione

- Il sito

  www.99-bottles-of-beer.net

  decrive come programmare in 1500 linguaggi di programmazione il testo di "*99 Bottles of Beer*"

Evoluzione dei linguaggi

1970	2014	2014	2014
Fortran	C	Java	JavaScript
Lisp	Java	PHP	Ruby
Cobol	Objective-C	Python	Java
Algol 60	C++	C#	Python
APL	C#	C++	PHP
Snobol 4	PHP	C	C
Simula 67	Visual Basic	JavaScript	C++
Basic	Python	Objective-C	CCS
PL/1	JavaScript	Ruby + Rails	C#
Pascal	Transact-SQL	Visual Basic	Objective-C
	TIOBE Index January 2014	PYPL Index January 2014	GitHub Repositories January 2014

Un po' di storia dei linguaggi di programmazione

Linguaggi di programmazione

I linguaggi di programmazione nascono con la macchina di Von Neumann (macchina a programma memorizzato)

- i programmi sono un particolare tipo di dato rappresentato nella memoria della macchina
- la macchina possiede un interprete capace di eseguire il programma memorizzato, e quindi di implementare ogni algoritmo descrivibile nel "linguaggio macchina"
- un linguaggio macchina dotato di semplici operazioni primitive per la scelta e per iterare (o simili) è Turing-equivalente, cioé può descrivere tutti gli algoritmi

Anni ‘50

- FORTRAN e COBOL (sempreverdi)
  - notazioni **simboliche** orientate rispettivamente al calcolo scientifico (numerico) e alla gestione dati (anche su memoria secondaria)
  - **astrazione procedurale** (sottoprogrammi, ma con caratteristiche molto simili ai costrutti forniti dai linguaggi macchina)
  - **Meccanismi linguistici** per introdurre **nuove operazioni e strutture dati** (per esempio, gli array in FORTRAN e i record in COBOL)
  - All’occhio moderno: nulla di significativamente diverso dai linguaggi macchina

I favolosi ‘60: LISP e ALGOL

- ✓ **Fondamenti** (teoria)
  - ✓ formalizzazione degli aspetti sintattici
  - ✓ primi risultati semantici basati sul lambda-calcolo
- ✓ **Caratteristiche comuni**
  - ✓ introduzione della nozione di ambiente per la gestione degli identificatori e le regole di scope
  - ✓ vera astrazione procedurale con ricorsione
- ✓ **ALGOL 60**
  - ✓ primo linguaggio imperativo veramente ad alto livello
  - ✓ scoping statico e gestione dinamica della memoria a stack
- ✓ **LISP** (*sempreverde*)
  - ✓ primo linguaggio funzionale, direttamente ispirato al lambda-calcolo (la teoria ritorna)
  - ✓ scoping dinamico, strutture dati dinamiche, gestione dinamica della memoria a heap con garbage collector

- ***ALGOL 60, prototipo dei linguaggi imperativi***
- ***LISP, prototipo dei linguaggi logici e funzionali***

- Analizzando i due linguaggi ci accorgiamo che originano concetti simili non a caso basati sulla teoria
  - La gestione dell'ambiente tramite lo stack
- gli approcci restano diversi e originano due filoni
  - *il filone imperativo (esempio C)*
  - *il filone funzionale (esempio OCaml)*

La fine degli anni ‘60

- **PL/I**: primo tentativo di linguaggio “totalitario” (targato IBM)
  - tentativo di sintesi fra LISP, ALGOL 60 e COBOL
  - fallito per mancanza di una visione semantica unitaria
- **SIMULA 67**: nasce di fatto la *programmazione a oggetti*
  - estensione di ALGOL 60 orientato alla simulazione discreta
  - quasi sconosciuto, riscoperto 15 anni dopo

Evoluzione del filone imperativo

- Gli anni ’70
  - metodologie di programmazione, tipi di dati astratti, modularità, classi e oggetti
  - programmazione di sistema in linguaggi ad alto livello: eccezioni e concorrenza
- Un esempio: **PASCAL**
  - estensione di ALGOL 60 con definizione di tipi (non astratti), uso esplicito di puntatori e gestione dinamica della memoria a heap (senza garbage collector)
  - semplice implementazione mista (con P-Code, antesignano del bytecode), facilmente portabile

Il dopo PASCAL

- **C**: PASCAL + moduli + tipi astratti + eccezioni + interfaccia per interagire con il sistema operativo
- **ADA**: il secondo tentativo di linguaggio "totalitario" (targato US DoD)
  - C + concorrenza + costrutti per la programmazione in tempo reale
  - progetto ambizioso: grande enfasi su semantica statica (proprietà verificabili dal compilatore)
- **C++**: C + classi e oggetti (allocati sullo heap, ancora senza garbage collector)

La programmazione logica

PROLOG

- implementazione di un frammento del calcolo dei predicati del primo ordine (la teoria che aiuta)
- strutture dati molto flessibili (termini) con calcolo effettuato dall'algoritmo di unificazione
- computazioni non-deterministiche
- gestione memoria a heap con garbage collector

CLP (Constraint Logic Programming)

- PROLOG + calcolo su domini diversi (anche numerici) con opportuni algoritmi di soluzione di vincoli

La programmazione funzionale

ML: implementazione del lambda-calcolo tipato

- definizione di nuovi tipi ricorsivi, i valori dei nuovi tipi sono termini, che possono essere visitati con un meccanismo di pattern matching (versione semplificata dell'unificazione)
- scoping statico (a differenza di LISP)
- semantica statica molto potente (inferenza e controllo dei tipi)
  - un programma "corretto" per la semantica statica quasi sempre va bene
- gestione memoria a heap con garbage collector

HASKELL= ML con regola di valutazione "lazy"

JAVA

- Molte caratteristiche dal filone imperativo
  - essenzialmente tutte quelle di C++
- alcune caratteristiche dei linguaggi logico-funzionali
  - gestione della memoria con garbage collector
- utilizza il meccanismo delle classi e dell'ereditarietà per ridurre il numero di meccanismi primitivi
  - quasi tutto è realizzato con classi predefinite nelle librerie
- ha una implementazione mista, tipica del filone logico
  - che ne facilita la portabilità e lo rende molto adatto ad essere integrato nelle applicazioni di rete

C#

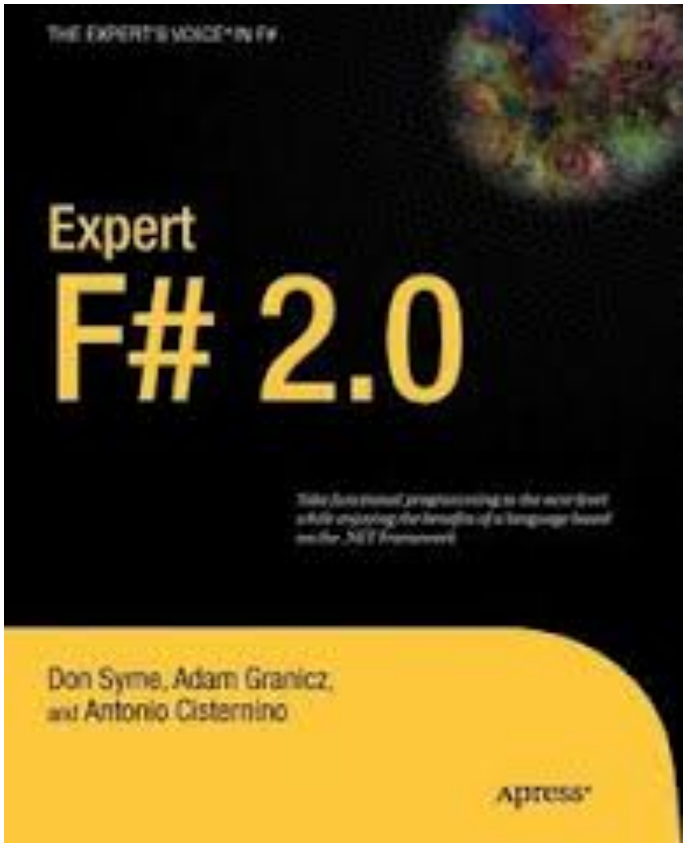
- C#: linguaggio di programmazione a oggetti sviluppato per la programmazione nel framework .NET
  - il “meglio” di Java e C++
- I tipi primitivi del linguaggio hanno una corrispondenza precisa con i tipi disponibili a run-time

SCALA

- Scala smoothly integrates features of object-oriented and functional languages

F#

- ML spiegato al popolo



Evoluzione dei linguaggi

a. Un ecosistema di applicazioni differenti
b. Enfasi crescente sulle astrazioni per il programmatore
c. Caratteristiche significative: migliorare la affidabilità, la manutenibilità e la sicurezza del software
d. Aspetti moderni: astrazioni per mobilità e distribuzione
e. Primitive linguistiche e astrazioni per parallelismo e concorrenza
f. Trend: *multi-paradigm programming*

Un esempio

- **Python** linguaggio di programmazione sviluppato a fine anni ‘80 da Guido van Rossum (CWI)
- Uno “scripting language”
- Linguaggio multi-paradigma: supporta in modo nativo oggetti e funzioni di ordine superiore tipiche della programmazione funzionale
- Tipi dinamici e gestione dinamica della memoria

Ruby

- **Ruby** linguaggio di scripting sviluppato a fine anni ’90 da Yukihiro Matsumoto
- Influenzato da Perl and Smalltalk
- Multi-paradigma: funzionale, a oggetti, imperativo con meccanismi di meta-programmazione (LISP che ritorna)
- Ruby (come lo descrivono)
  - *everything is an object*
  - *every operation is a method call*
  - *all programming is meta-programming*
- Usato nello sviluppo di applicazioni web

Paradigma funzionale per Java e C#

- Java 8: la versione corrente di Java
- Introduzione di meccanismi linguistici per la programmazione funzionale: **Lambda**
  - Problema: introdurre Lambda senza dover ricompilare i codici binari esistenti.
- Espressioni Lambda sono disponibili anche in C#
  - …con il medesimo scopo

Modelli computazionali

- Come vedremo meglio nella seconda parte del corso a ogni linguaggio è associato un ***modello di calcolo***
  - **Imperativo**: Fortran (1957)
  - **Funzionale**: Lisp (1958)
  - **A oggetti**: Simula (1967)
  - **Logico**: Prolog (1972)
  - **Relazionale** : SQL (1974)

Il progetto di PR2

- Un metodo efficace per comprendere cosa significa “modello di computazione” è progettare e sviluppare un linguaggio di programmazione
  - Il progetto di PR2 si propone questo obiettivo!!