VIA University College

Programming Concepts and Languages

Spring 2024

Agenda

- ✓ Introduction
- ✓ Why learn alternative paradigms & languages?
- ✓ Principal programming paradigms
- ✓ Why F# and Python?
- ✓ What is Functional Programming?
- ✓ F# on Visual Studio Code
- ✓ Exercises

Who am I ?

Learning Objectives

- By the end of this course, you will be able to:
  - identify the various programming paradigms and describe their strengths and weaknesses.
  - explain fundamental programming concepts in the different programming paradigms
  - apply fundamental programming concepts, using multi-paradigm programming languages, to solve substantial problems
  - develop, implement, and test simple programs and applications using the four major programming paradigms.

Course Style

- Everything will be on itslearning:
  - Lessons and exercises in class
  - Approx. 12 weeks, 48 lessons.
- Notes:
  - Lecture notes will be available on the course web-page
- Exercises, course-project and presentations during the course
- Grading/Examination
  - Three-hour written examination - 100%

Learning Style

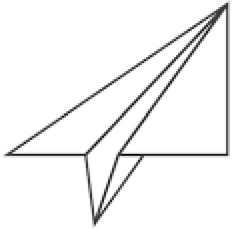
- Lessons
  - Class exercises

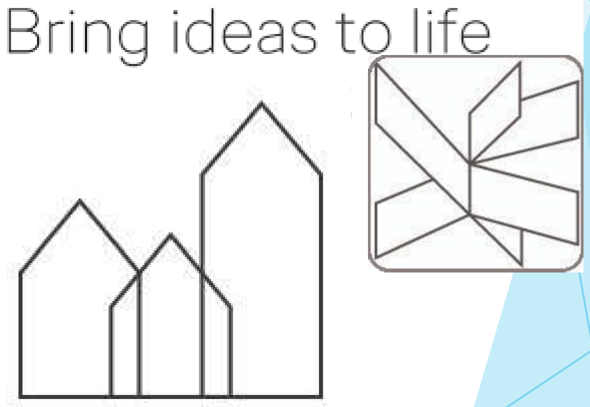
- Project
  - Course project group (2 - 3 persons)
    - **Deadline: Please keep to deadlines**

- …

Course-project

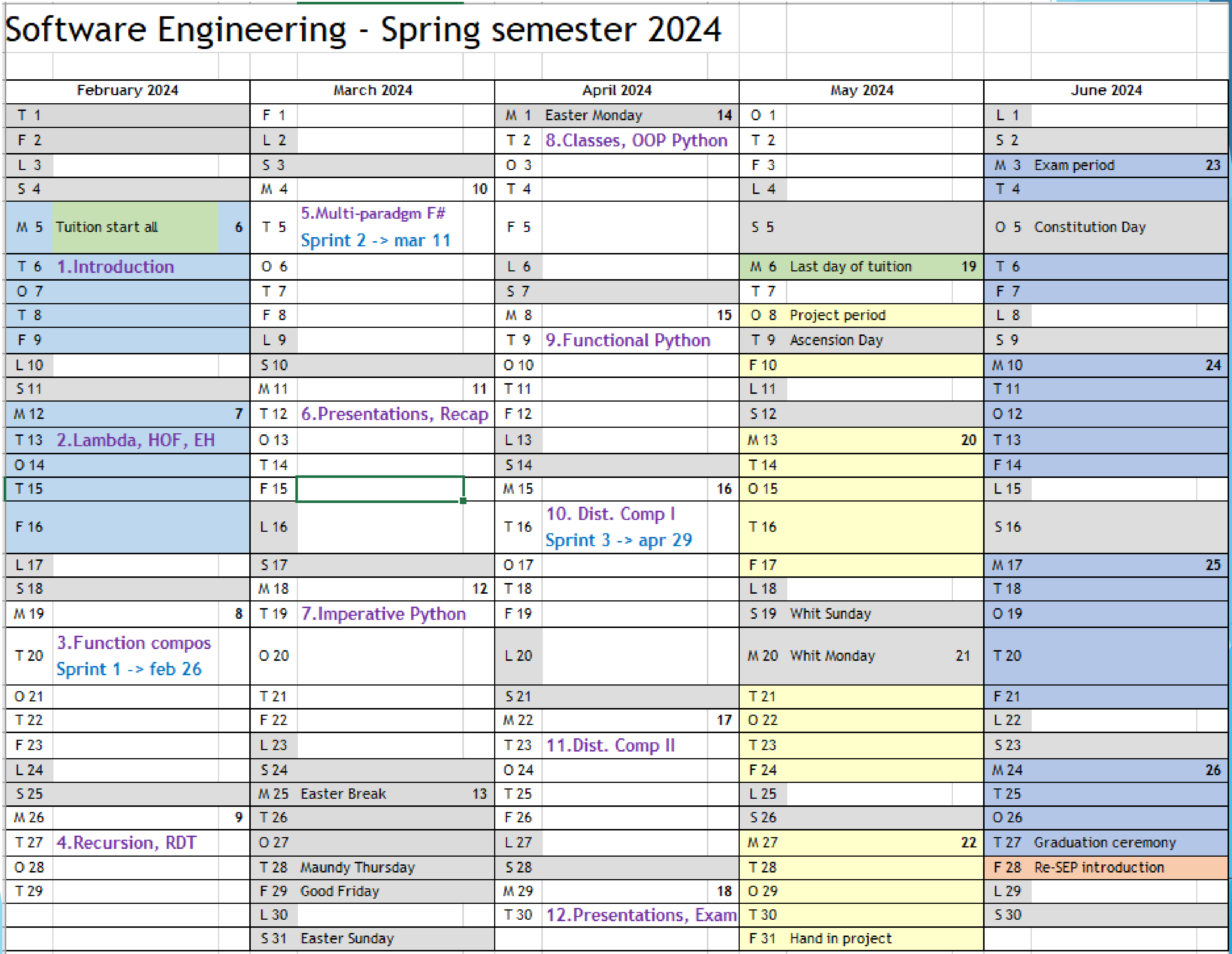
- Course-project includes implementation of some of the different concepts in the different paradigms:
  - ✓ * functional
  - ✓ * concurrent
  - ✓ * distributed
  - ○ + object-oriented
  - ○ + imperative





```
getAllStudents().filter(s => s.semester > 6).sort()
|> allocateProjectRoom() |>  haveFun()
```

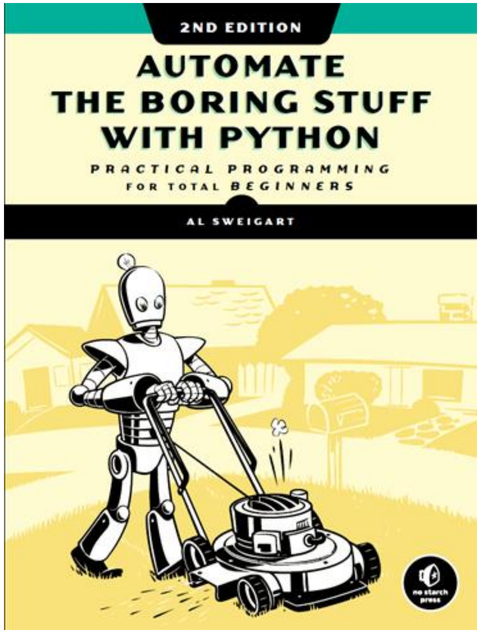
Tentative plan

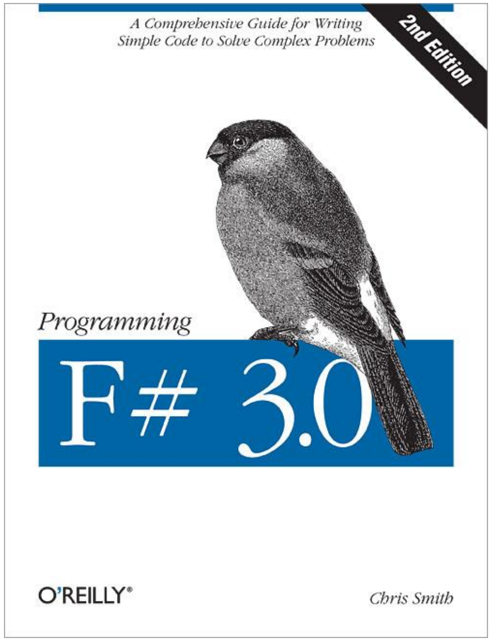


Software Engineering - Spring semester 2024

February 2024			March 2024			April 2024			May 2024			June 2024		
T 1			F 1			M 1	Easter Monday	14	O 1			L 1		
F 2			L 2			T 2	8.Classes, OOP Python		T 2			S 2		
L 3			S 3			O 3			F 3			M 3	Exam period	23
S 4			M 4		10	T 4			L 4			T 4		
M 5	Tuition start all	6	T 5	5.Multi-paradgm F# Sprint 2 -> mar 11		F 5			S 5			O 5	Constitution Day	
T 6	1.Introduction		O 6			L 6			M 6	Last day of tuition	19	T 6		
O 7			T 7			S 7			T 7			F 7		
T 8			F 8			M 8		15	O 8	Project period		L 8		
F 9			L 9			T 9	9.Functional Python		T 9	Ascension Day		S 9		
L 10			S 10			O 10			F 10			M 10		24
S 11			M 11		11	T 11			L 11			T 11		
M 12		7	T 12	6.Presentations, Recap		F 12			S 12			O 12		
T 13	2.Lambda, HOF, EH		O 13			L 13			M 13		20	T 13		
O 14			T 14			S 14			T 14			F 14		
T 15			F 15			M 15		16	O 15			L 15		
F 16			L 16			T 16	10. Dist. Comp I Sprint 3 -> apr 29		T 16			S 16		
L 17			S 17			O 17			F 17			M 17		25
S 18			M 18		12	T 18			L 18			T 18		
M 19		8	T 19	7.Imperative Python		F 19			S 19	Whit Sunday		O 19		
T 20	3.Function compos Sprint 1 -> feb 26		O 20			L 20			M 20	Whit Monday	21	T 20		
O 21			T 21			S 21			T 21			F 21		
T 22			F 22			M 22		17	O 22			L 22		
F 23			L 23			T 23	11.Dist. Comp II		T 23			S 23		
L 24			S 24			O 24			F 24			M 24		26
S 25			M 25	Easter Break	13	T 25			L 25			T 25		
M 26		9	T 26			F 26			S 26			O 26		
T 27	4.Recursion, RDT		O 27			L 27			M 27		22	T 27	Graduation ceremony	
O 28			T 28	Maundy Thursday		S 28			T 28			F 28	Re-SEP introduction	
T 29			F 29	Good Friday		M 29		18	O 29			L 29		
			L 30			T 30	12.Presentations, Exam		T 30			S 30		
			S 31	Easter Sunday					F 31	Hand in project				

Course Material


2ND EDITION
AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR TOTAL BEGINNERS
AL SWEIGART
no starch press


A Comprehensive Guide for Writing Simple Code to Solve Complex Problems
2nd Edition
Programming
F# 3.0
O'REILLY®
Chris Smith

- https://docs.python.org/3/
- https://peps.python.org/pep-0008/
- Others: Links, papers and videos about the general programming concepts and languages, Python programming websites

Why learn alternative paradigms & languages?

- Because it will:
  - broaden the way you think about programming, and help you solve problems in new ways.
  - prepare you for future paradigms & languages.
  - help you in comparing languages & paradigms.
  - help you understand languages at a deeper level, instead of just the syntax.

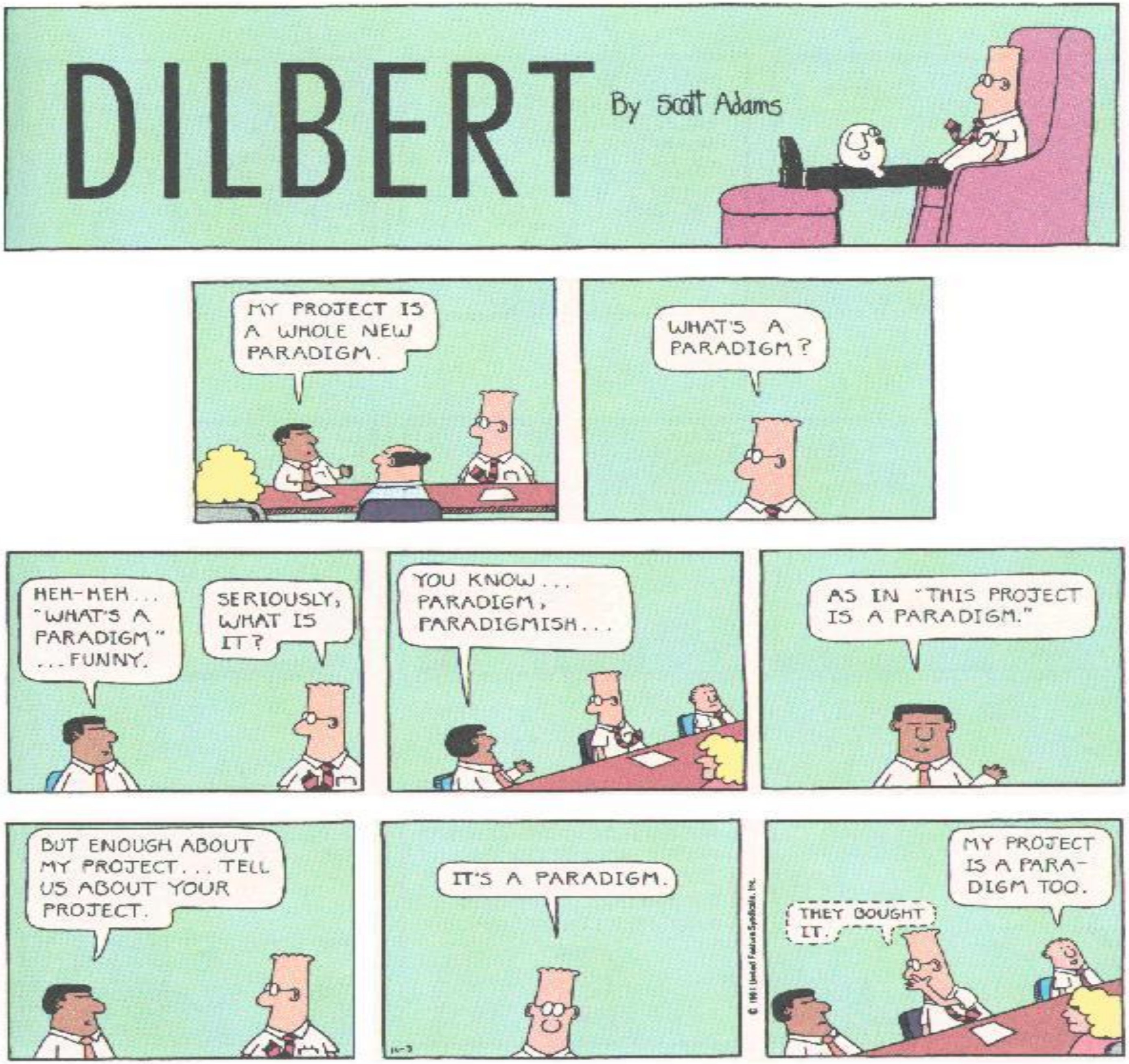
What should we learn?

- different programming paradigms
- and languages that will determine the suitable language to a particular problem or project.

What We Learn becomes a part of who we are
ORIGINAL WORDS + DESIGN © 2013, KATHY R. JEFFORDS

`as software engineers/programmers`

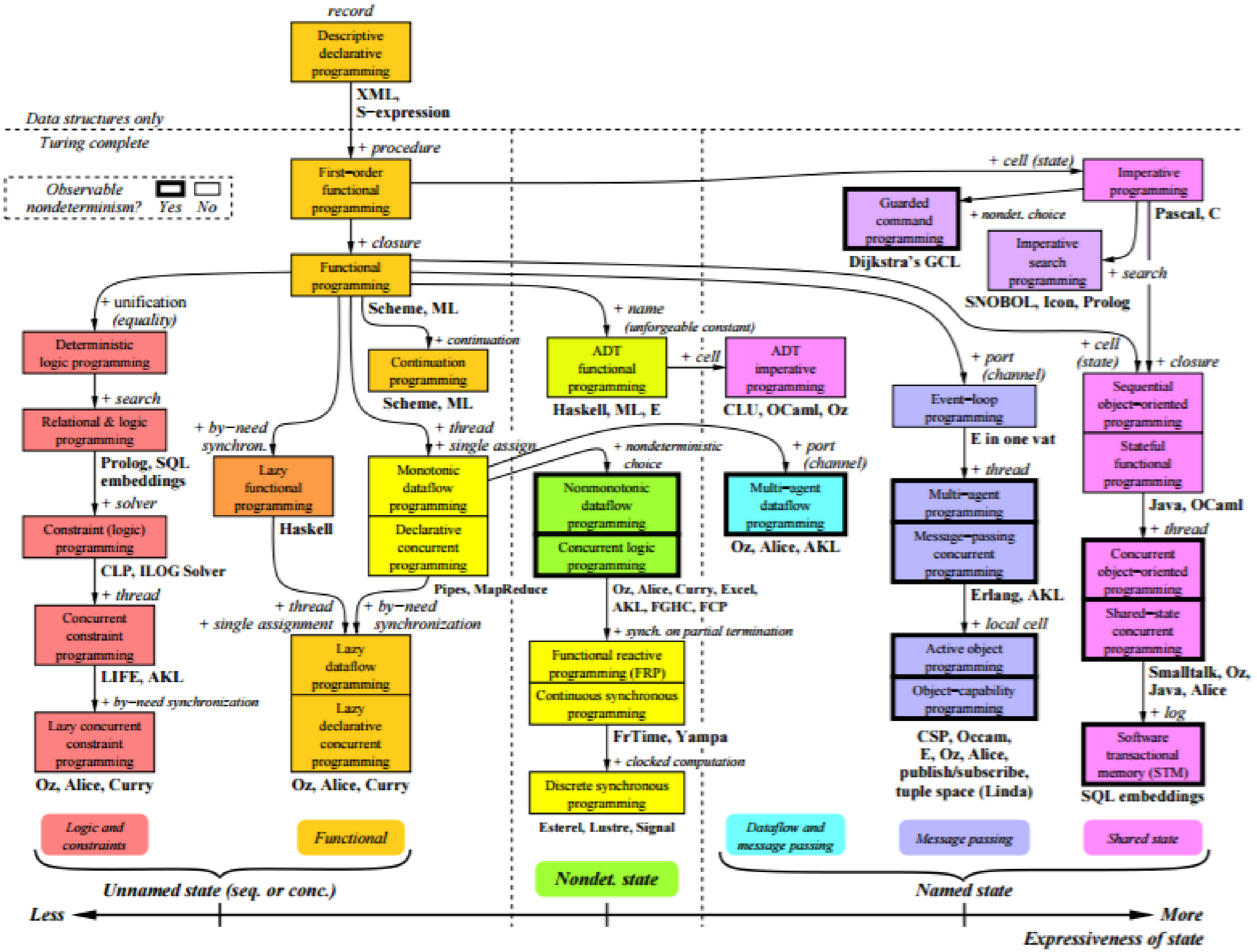
Knowing different paradigms and languages is part of the job of being a professional software engineer


DILBERT
By Scott Adams
MY PROJECT IS A WHOLE NEW PARADIGM.
WHAT'S A PARADIGM?
HEH-HEH... "WHAT'S A PARADIGM" ...FUNNY.
SERIOUSLY, WHAT IS IT?
YOU KNOW... PARADIGM, PARADIGMISH...
AS IN "THIS PROJECT IS A PARADIGM."
BUT ENOUGH ABOUT MY PROJECT... TELL US ABOUT YOUR PROJECT.
IT'S A PARADIGM.
THEY BOUGHT IT.
MY PROJECT IS A PARADIGM TOO.
11-3
© 1991 United Feature Syndicate, Inc.

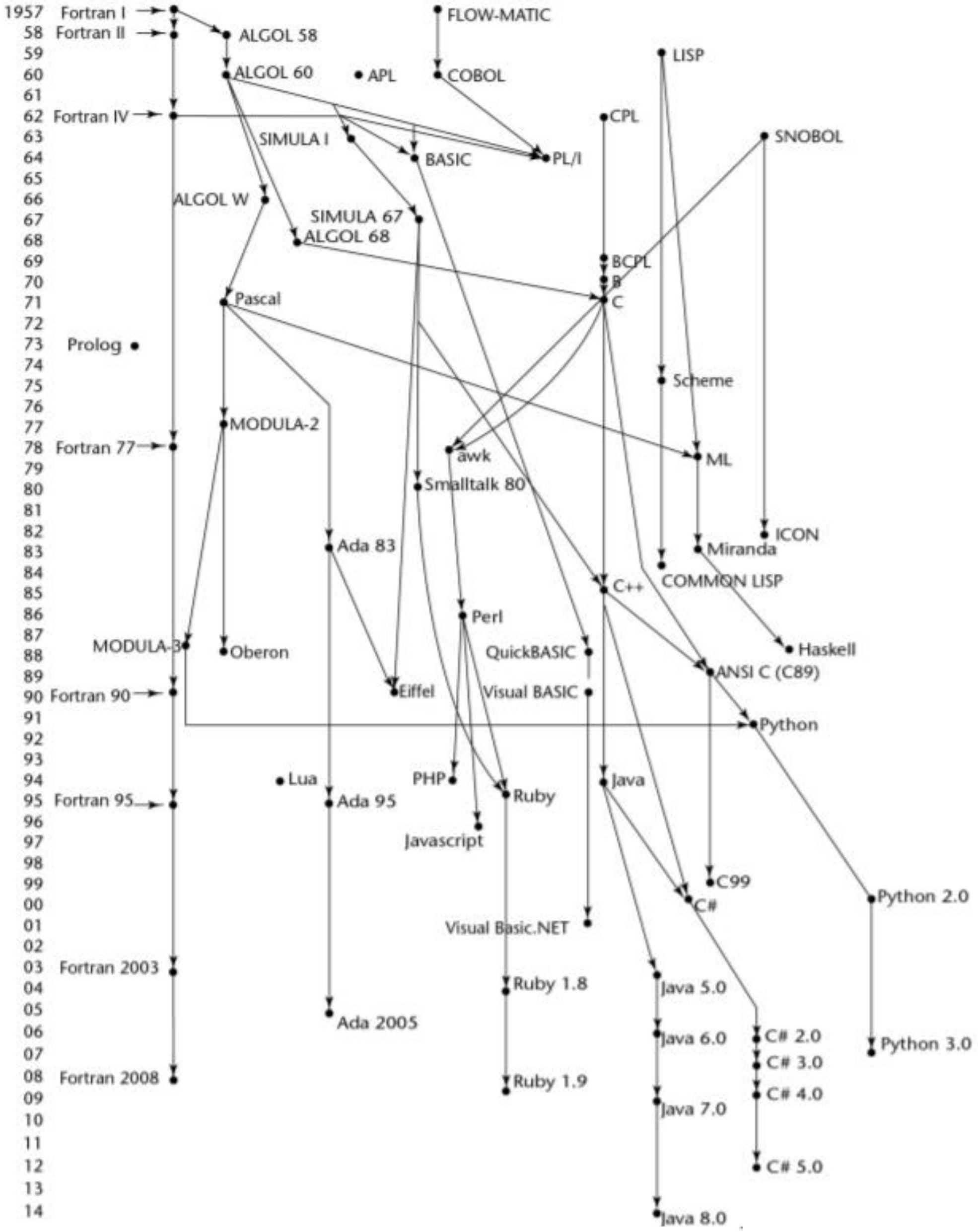
Definitions

- Programming Language
  - notation for specifying programs/computations
  - consists of words, symbols, and rules for writing a program
- Programming Paradigm
  - programming “technique”
  - way of thinking about programming
  - view of a program
  - Patterns that serves as “school of thoughts” for programming

Taxonomy of Programming paradigms


record
Descriptive declarative programming
XML, S−expression
Data structures only
Turing complete
+ procedure
Observable nondeterminism? Yes No
First−order functional programming
+ cell (state)
Imperative programming
Guarded command programming
+ nondet. choice
Pascal, C
Dijkstra's GCL
Imperative search programming
+ search
SNOBOL, Icon, Prolog
+ closure
Functional programming
+ unification (equality)
Scheme, ML
+ name (unforgeable constant)
+ continuation
Deterministic logic programming
Continuation programming
ADT functional programming
+ cell
ADT imperative programming
+ port (channel)
+ cell (state)
+ closure
Scheme, ML
Haskell, ML, E
CLU, OCaml, Oz
Event−loop programming
Sequential object−oriented programming
+ search
Relational & logic programming
+ by−need synchron.
+ thread
+ single assign.
E in one vat
Stateful functional programming
+ nondeterministic choice
+ port (channel)
Prolog, SQL embeddings
Lazy functional programming
Monotonic dataflow programming
Nonmonotonic dataflow programming
Multi−agent dataflow programming
+ thread
Multi−agent programming
Java, OCaml
+ solver
Declarative concurrent programming
Concurrent logic programming
+ thread
Constraint (logic) programming
Haskell
Oz, Alice, AKL
Message−passing concurrent programming
Concurrent object−oriented programming
CLP, ILOG Solver
Pipes, MapReduce
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
Erlang, AKL
+ thread
+ thread
+ single assignment
+ by−need synchronization
Shared−state concurrent programming
Concurrent constraint programming
+ synch. on partial termination
+ local cell
Lazy dataflow programming
Functional reactive programming (FRP)
Active object programming
Smalltalk, Oz, Java, Alice
LIFE, AKL
+ by−need synchronization
Continuous synchronous programming
Object−capability programming
+ log
Lazy concurrent constraint programming
Lazy declarative concurrent programming
FrTime, Yampa
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
Software transactional memory (STM)
+ clocked computation
Oz, Alice, Curry
Oz, Alice, Curry
Discrete synchronous programming
SQL embeddings
Esterel, Lustre, Signal
Logic and constraints
Functional
Dataflow and message passing
Message passing
Shared state
Nondet. state
Unnamed state (seq. or conc.)
Named state
Less
More
Expressiveness of state

https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf


1957 Fortran I
FLOW-MATIC
58 Fortran II
ALGOL 58
59
LISP
60
ALGOL 60
APL
COBOL
61
62 Fortran IV
CPL
63
SIMULA I
SNOBOL
64
BASIC
PL/I
65
66
ALGOL W
67
SIMULA 67
68
ALGOL 68
69
BCPL
70
B
71
Pascal
C
72
73
Prolog
74
75
Scheme
76
77
MODULA-2
78 Fortran 77
awk
ML
79
80
Smalltalk 80
81
82
ICON
83
Ada 83
Miranda
84
COMMON LISP
85
C++
86
Perl
87
Haskell
88
MODULA-3
Oberon
QuickBASIC
ANSI C (C89)
89
90 Fortran 90
Eiffel
Visual BASIC
91
Python
92
93
94
Lua
PHP
Java
95 Fortran 95
Ada 95
Ruby
96
97
Javascript
98
99
C99
00
Python 2.0
C#
01
Visual Basic.NET
02
03 Fortran 2003
04
Ruby 1.8
Java 5.0
05
Ada 2005
06
Java 6.0
C# 2.0
Python 3.0
07
C# 3.0
08 Fortran 2008
Ruby 1.9
09
C# 4.0
Java 7.0
10
11
12
C# 5.0
13
14
Java 8.0

Fortan

```
PROGRAM WELCOMEPCL
10 FORMAT (1X,14HWELCOME TO PCL)
WRITE(6,10)
END
```

Basic

```
PRINT “Welcome to PCL!"
```

C

```
#include

int main(void)
{
    puts(“Welcome to PCL!");
}
```

C#

```
using System;
class Program
{
  public static void Main(string[] args)
  {
    Console.WriteLine(“Welcome to PCL!");
  }
}
```

Python

```
print("Welcome to PCL!“)
```

Algol-60

```
begin
      file rmt (kind = remote);
      write(rmt, <“Welcome to PCL!">);
 end.
```

C++

```
#include

int main()
{
    std::cout << “Welcome to PCL!
";
    return 0;
}
```

Java

```
public class WelcomeToPCL {
  public static void main(String []args)
  {
    System.out.println(“Welcome to PCL");
        }
  }
```

JavaScript

```
document.writeln("Welcome to PCL!");
```

R

```
cat('Welcome to PCL!
')
```

Why so many?

- Most important: the choice of paradigm (and therefore language) depends on how humans best think about the problem
- Other considerations:
  - efficiency
  - compatibility with existing code
  - availability of translators

What next?

- Once you've understood the general concepts of programming paradigms, learning new programming languages becomes easier.
- N/B Picking the right paradigm does not solve all the problems.
- As noted by Flon
  - *"There does not now, nor will there ever exist, a programming language in which it is the least bit hard to write bad programs." L. Flon*

Principal Programming Paradigms

- Imperative Programming
  - program as a collection of statements and procedures affecting data (variables)
- Object-Oriented Programming
  - program as a collection of classes for interacting objects
- Functional Programming
  - program as a collection of (math) functions
- Others
  - Concurrent
    - Allows many things to happen concurrently
  - Distributed
    - Allows for inter process communication and applications to be separated into smaller parts
    - Message System –Publish/Subscribe, Microservices

Imperative Programming I

- Variables, assignment, sequencing, iteration, procedures as units
- State-based, assignment-oriented
- Global variables, side effects

- Program units: Data (Variables) and Computation (Statements and Routines)

Imperative Programming II

- Imperative Programming is about
  - Data (variables) and statements affecting that data
  - Control-flow constructs enrich statement specification
  - Routines and modules help impose program organization
- Advantages
  - Low memory utilization
  - Relatively efficient
  - The most common form of programming in use today
- Disadvantages
  - Difficulty in parallelization
  - Tend to be relatively low level
  - Difficulty in reasoning about programs

Object-Oriented Programming I

- Incorporates both encapsulation and inheritance through the class concept
- Focus is on writing good classes and on code reuse
- Examples
  - Shape, Circle, and Rectangle in a drawing program
  - Employee, Faculty, Staff in a university personnel system

Object-Oriented Programming II

- Program consists of a collection of objects that interact by passing messages that transform object state.
- OO languages are characterized by
  - Data encapsulation/abstraction
  - Inheritance
  - Polymorphism
- Advantages
  - Conceptual simplicity
  - Models computer better
  - Increased productivity
- Disadvantages
  - Doing I/O can be cumbersome
  - A bit of initial steep learning curve

Imperative vs Object-Oriented

- Imperative programs consists of actions to effect state change, principally through assignment operations or side effects
  - Fortran, Algol, Cobol, Pascal, C
- Object-Oriented programming is not always imperative, but most OO languages have been imperative
  - C++, Java

Functional Programming I

- Functional programming models a computation as a collection of mathematical functions.
  - Input = domain
  - Output = range
- Functional languages are characterized by:
  - Functional composition
  - Recursion
- Focuses on function evaluation; avoids updates, assignment, mutable state, side effects
- Not all functional languages are “pure”
  - In practice, rely on non-pure functions for input/output and some permit assignment-like operators

Functional Programming II

- Program execution involves functions calling each other and returning results. There are no variables in functional languages
- Advantages
  - Small and clean syntax
  - Better support for reasoning about programs
  - They allow functions to be treated as any other data values.
  - Supports programming at a relatively higher level than the imperative languages.
- Disadvantages
  - Difficulty of doing input-output
  - Functional languages use more storage space than their imperative cousins

Concurrent Programming

- Concurrent programming cuts across imperative, object-oriented, and functional paradigms

- Concurrent programming = spawn independent processes, which live independent lives

Principal Programming Paradigms and Languages

- Imperative Programming
  - program as a collection of statements and procedures affecting data
  - **Python** , **F#**, FORTRAN, BASIC, COBOL, Pascal, C, etc
- Object-Oriented Programming
  - program as a collection of classes for interacting objects
  - **Python** , Scala , C#, Java, SmallTalk, etc
- Functional Programming
  - program as a collection of (math) functions
  - **F#** *, **Python** *, R, LISP, ML, Haskell, etc.
- Others: Concurrent, Microservices
  - allows applications to be separated into smaller parts.
  - **F# , Python**, Java, C#, etc.

* In reality, very few languages are "pure"

  - Most combine features of different paradigms

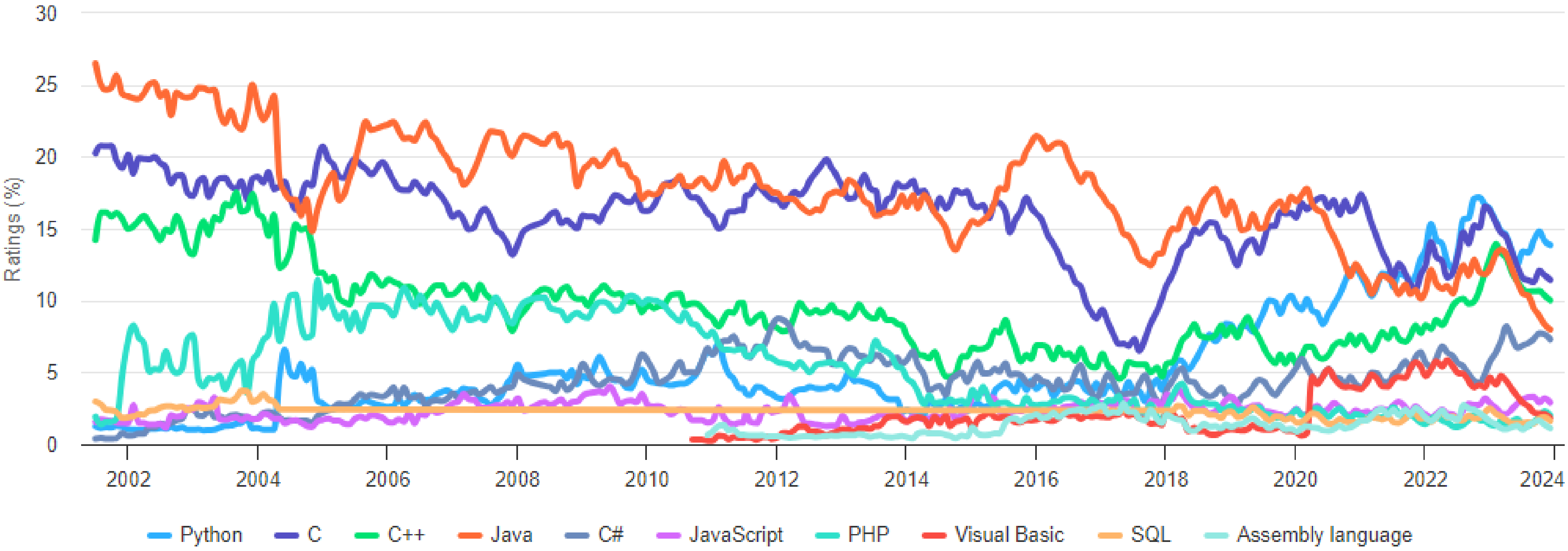
Why Emphasis on Functional Programming?

- Functional programming is one of the oldest paradigms.
  - Lisp originated in 1958!
  - It is still widely used and has been highly influential.
- Functional programming can be very elegant and has a strong mathematical foundation.
- Many other paradigms can be neatly interpreted in terms of functional programming.
- It is the “next big thing”.
  - Cloud Functions
  - AWS LAMBDA

Why F# and Python?

- F#:
  - Concise with functional first paradigm - awareness of side effects
  - functions as first-class citizen, discriminated unions
  - interop with .NET ecosystem
  - DDD - Domain Driven Development
- Python:
  - versatile, flexible, and object-oriented features
  - loved by developers, data scientists, software engineers, etc
  - Python on track to be crowned 'programming language of the year'. Ref. TIOBE, Jan 2023.
- … and more

Python is TIOBE's programming language of the year 2023!



https://www.tiobe.com/tiobe-index/

TIOBE Index for December 2023

(Python is the language of the year 2023 and #1)

Review

- Which of these paradigms is the best?

A quick look at some concepts

- Unifying language concepts
  - Types (both built-in and user-defined)
    - Specify constraints on functions and data
    - Static vs. dynamic typing
  - Expressions (e.g., arithmetic, boolean, strings)
  - Functions

Language Translation

- **Native-code compiler**: produces machine code
  - Compiled languages: C, C++
- **Interpreter**: translates into internal form and immediately executes (read-eval-print loop)
  - Interpreted languages: Scheme, Haskell, Python …
- **Byte-code compiler**: produces portable bytecode, which is executed on virtual machine (e.g., Java)
- Hybrid approaches
  - Source-to-source translation (early C++ → C→compile)
  - Just-in-time Java compilers convert bytecode into native machine code when first executed

Language Compilation

- Compiler: program that translates a source language into a target language
    - Target language is often, but not always, the assembly language for a particular machine

Language interpretation

- Read-Eval-Print-Loop - REPL
  - Read in an expression, translate into internal form
  - Evaluate internal form
    - This requires an abstract machine and a “run-time” component (usually a compiled program that runs on the native machine)
  - Print the result of evaluation
  - Loop back to read the next expression

Bytecode Compilation

- Combine compilation with interpretation
  - Idea: remove inefficiencies of read-eval-print loop
- Bytecodes are conceptually similar to real machine opcodes, but they represent compiled instructions to a virtual machine instead of a real machine
  - Source code statically compiled into a set of bytecodes
  - Bytecode interpreter implements the virtual machine

Concept of Types

- A programming language needs to organize data in some way
- The constructs and mechanisms to do this are called type system
- Types become handy when:
  - designing programs
  - checking correctness
  - determining storage requirements

Type System

- The type system of a language usually includes
  - a set of predefined data types, e.g., integer, string
  - a mechanism to create new types, e.g., typedef
- mechanisms for controlling types:
  - equivalence rules: when are two types the same?
- compatibility rules: when can one type be substituted for another?
- inference rules: how is a type assigned to a complex expression?
- rules for checking types, e.g., static vs. dynamic

Static vs. Dynamic Typing I

- We also distinguish between languages depending on when they check typing constraints
- In static typing we check the types and their constraints before executing the program
  - Can be done during the compilation of a program
- When using dynamic typing, we check the typing during program execution

Static vs. Dynamic Typing II

- Static typing
  - Common in compiled languages, considered “safer”
  - Type of each variable determined at compile-time; constrains the set of values it can hold at run-time
  - + less error-prone
  - - sometimes too restrictive
- Dynamic typing
  - Common in interpreted languages
  - Types are associated with a variable at run-time; may change dynamically to conform to the type of the value currently referenced by the variable
  - Type errors not detected until a piece of code is executed
  - + more flexible
  - - harder to debug (if things go wrong)

Type inference

- The goal is to reconstruct types of expressions based on known types of some symbols that occur in expressions
- Best known in functional languages
  - Mostly used in managing the types of higher-order functions
- More details in the coming sessions

Which Programming Paradigm is Best?

- The accurate answer is that there is no best paradigm.
- No single paradigm will fit all problems well
- Use a combination of features represented by these paradigms