ABAP Development and Data Migration to SAP-HANA Concepts

Presented by,
Valliammai Radhakrishnan
Concepts of Programming Languages, Summer 2021.

AGENDA

- Introduction: ABAP
- Features of ABAP
- Migration to SAP-HANA
- ABAP to the future
- Conclusion

Introduction: ABAP

Advanced **B**usiness **A**pplication **P**rogramming –by SAP SE Germany in the year 1983.

Hybrid Program:

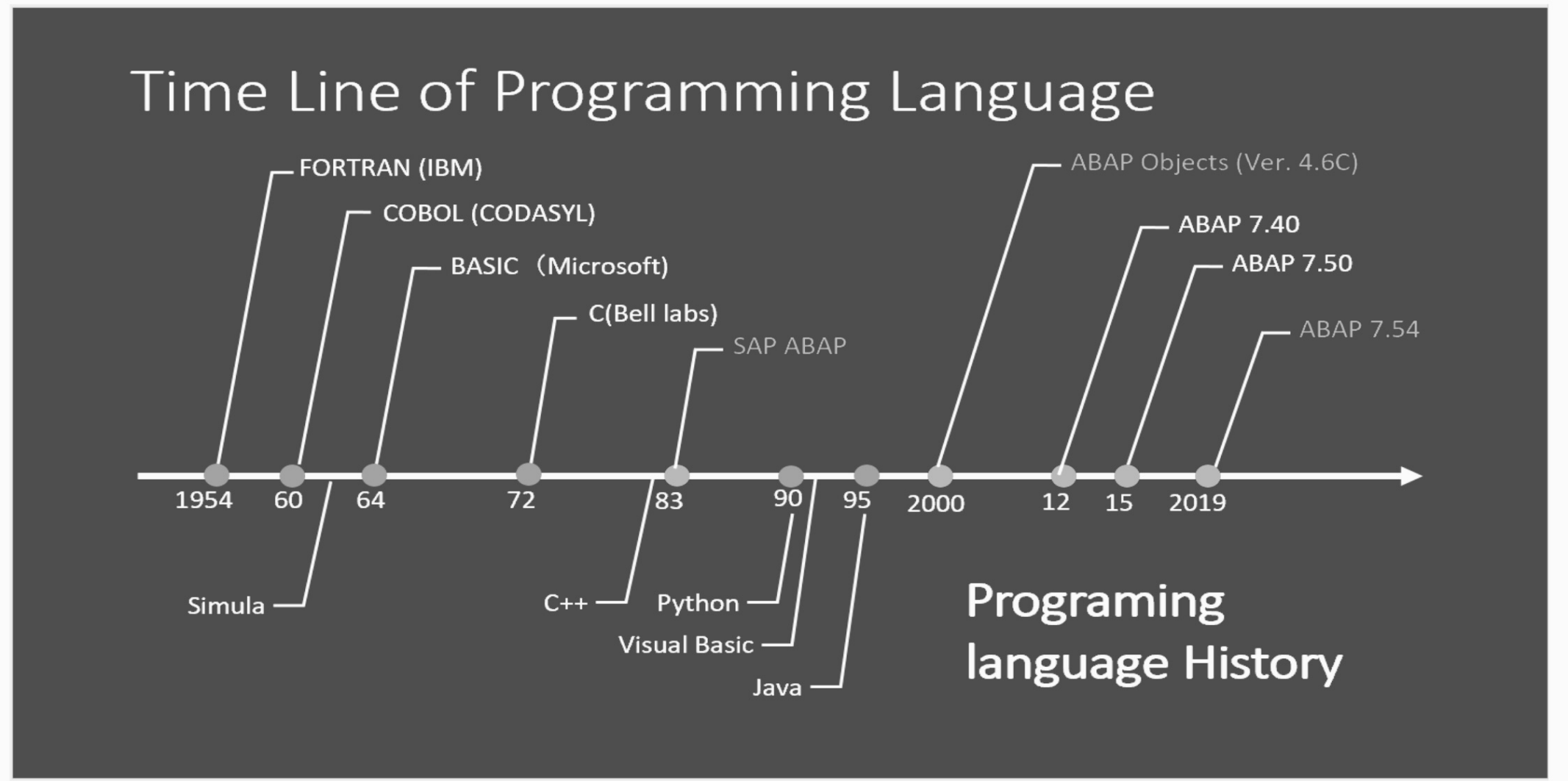
Procedural programming

Object Oriented Programming- ‘ABAP Objects’

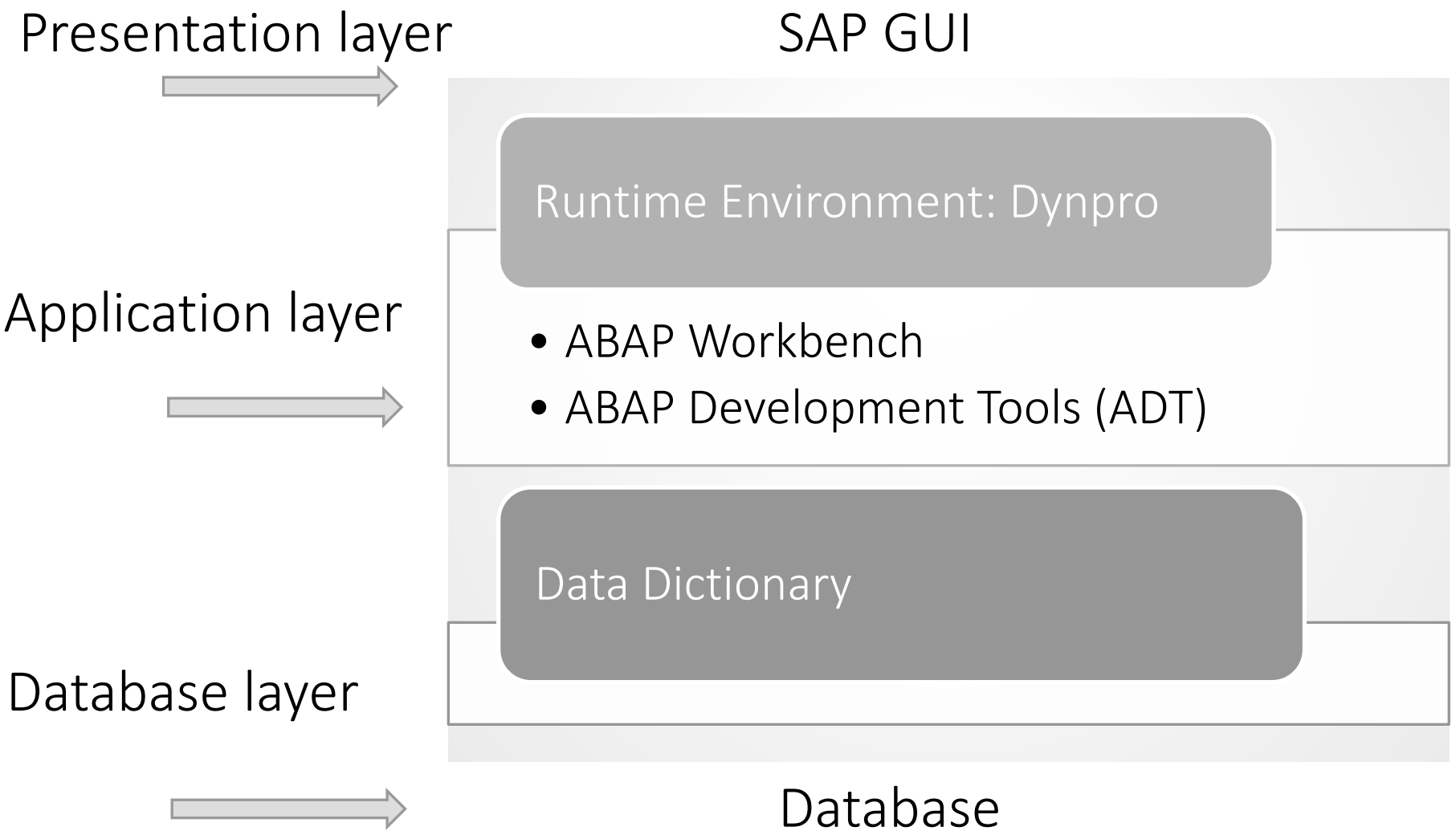
Aim: To solve complex business problems and large data management with simple set of codes.

Operational areas: Material management, Logistics, Production planning, and Sales data management.

Evolution of ABAP Objects



Where does ABAP lies in SAP NetWeaver?



Special features of ABAP

- Code based dynamic development
- Open SQL Statements
- Change Transport System (CTS)
- Internal Objects and tables
- Secured Infrastructure

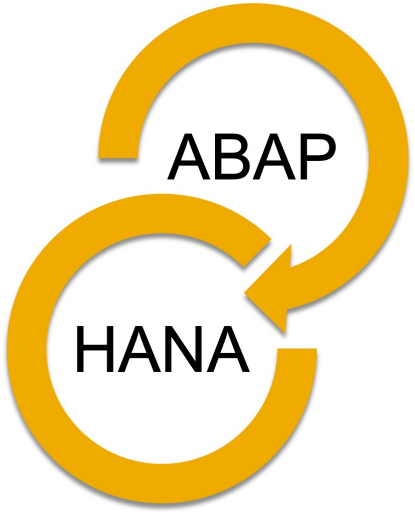
Add-ons:

- Core Data Service (CDS)
- De-pooling and De-clustering

Data Migration to SAP-HANA

SAP- High Performance Analytic Appliance (HANA) leaverages ABAP to the next level.

- In-memory
- Column Oriented
- Relational database system



Reason for Migration

- High speed processing
- Memory management using RAM

ABAP to the future

ABAP Restful Programming Model (RAP)

- ***New ABAP language features for more simple and concise code***
- ***ABAP application development further optimized for SAP HANA***
- ***Increased coverage of SQL standard***
- ***Efficient programming model for SAP Fiori applications***
- ***Industry 4.0 and big data scenarios support***
- ***Increased developer efficiency in Eclipse***
- ***Tool support for migration to SAP S/4HANA***

Conclusion

- SAP ABAP was created to run within the SAP Architecture as a tool to customise SAP business transactions and workflows.
- It made SAP, a world leader in terms of revenue.
- ABAP Object is extremely strong in terms of reflecting business logic and there is no other language so well adapted for this task.
- In conclusion, the popularity of a language doesn't equate to the language being superior.

References

1) A. Targowski and J. Tarn, “Enterprise systems education in the 21stcentury,”IGI Global, 2006.
2) P. Moxon, “Beginners guide to sap abap,”SAPPROUK Limited, 2012.
3) T. Leimbach, “The sap story: Evolution of sap within the german softwareindustry,” inIEEE Annals of the History of Computing, vol. 30, no. 4,2008, pp. 60–76.
4) May .N, Böhm .A, Lehner .W, “SAP HANA–The evolution of an inmemory DBMS from Pure OLAP processing towards mixed workloads”, Datenbanksysteme für Business, Technologie und Web (BTW 2017), 2017.
5) Pezzini M, Sholler .D, “SAP Throws Down the Next-Generation Architecture Gauntlet With HANA”, Gartner Research, Vol. 13, 2011.
6) Poschadel F. SAP S/4HANA: Performance Management in Real Time?. InPerformance Management in Retail and the Consumer Goods Industry 2019 (pp. 137-151). Springer, Cham.

- Thank you!

Assembler Programming Language

Concepts of Programming Languages, Summer 2021

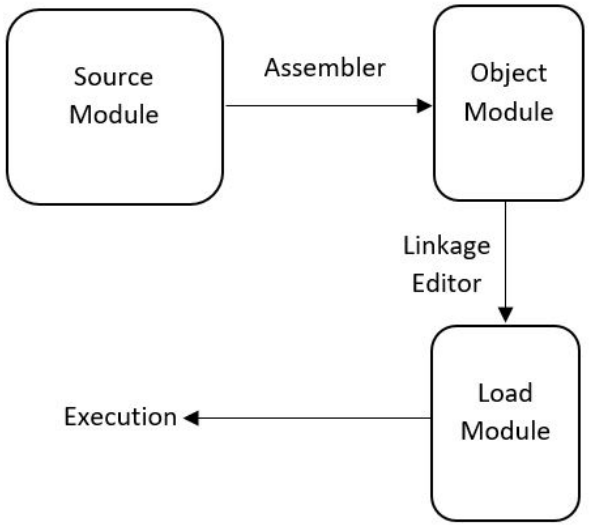
A presentation by Kaustubh Sarbhai

Agenda

- Introduction to Assembly Language
- Assembly Language elements
- The Assembly process
- How assembly language is different from High Level languages?
- Why we need to learn Assembly Language?
- Problems with Assembly Language
- Conclusion

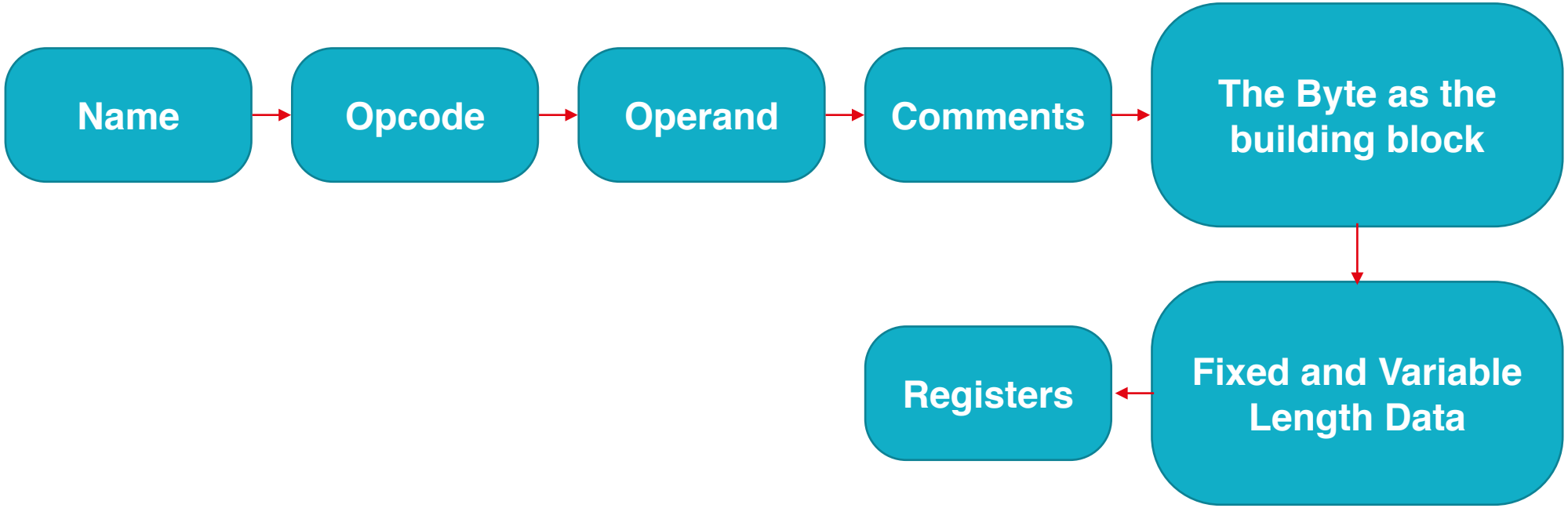
Introduction to Assembly Language

- Assembly language also known by symbolic machine code is one of the low level programming language which is still in use in the programming world.
- Assembly language allows developer to use mnemonics for referring a combination of 1s and 0s.
- Program code in assembly language get through three different stages before it gives final result.



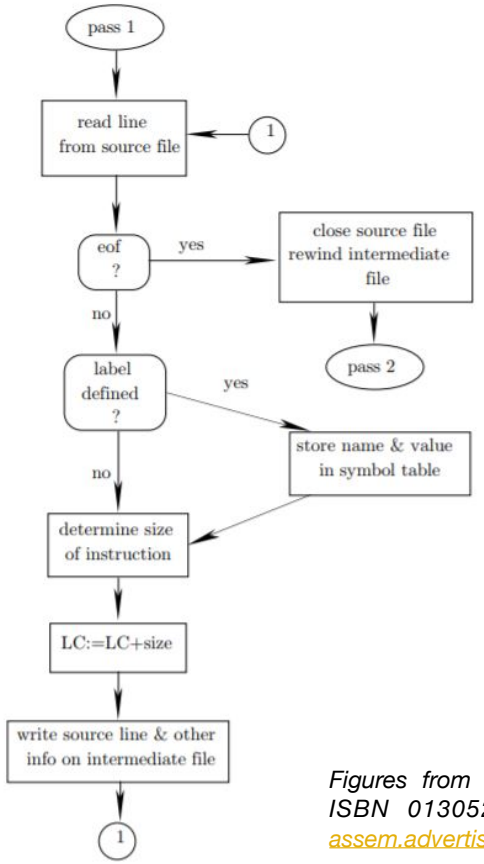
1. *Figure from* SharonK.Tuggle,*AssemblerLanguageProgramming:Systems/360and 370* , 1995.

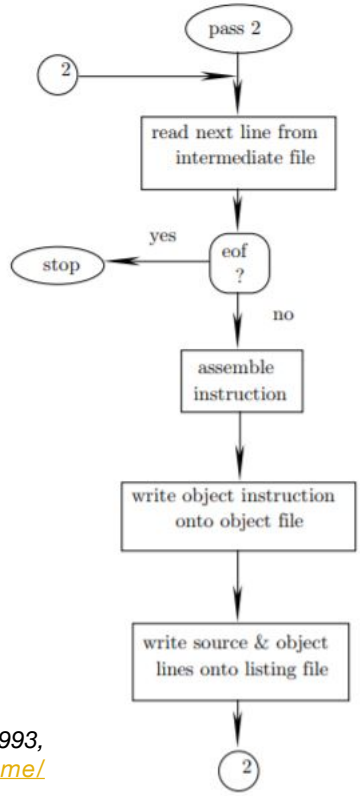
Assembly language elements



The Assembly Process

- The assembler allow to use only meaningful symbol to represent the location of an instruction.
- It also allocate storage for instructions and data and keep a track of it.
- Different assemblers follows different process for translating the source program to machine language.
- One such example is Two-Pass Assembler -





Figures from David Salomon, Assemblers And Loaders. 1993, ISBN 0130525642, URL: http://www.davidsalomon.name/assem.advertis/asl.pdf

How Assembly Language is different From High Languages?

- In high level languages we uses additional non transparent separation like variables.
- But there is no concept of variables in assembly language.
- In high level languages the programmer needs to follow the set of rules.
- In assembly programmer has freedom to do everything at any time.

Why we need to learn Assembly Language?

Short

Fast

Easy Learn

Problems with Assembly Language

- Hard to read
- Hard to debug
- Machine dependent
- Heavy documentation
- Time consuming
- No error checking

Conclusion

Questions?

An overview of C++

CPL

Summer Semester 2021

Konstantinos Loizas

Innovative Hochschule
Eine gemeinsame Initiative von Bund und Ländern

Agenda

- Introduction
- Rules of thumb
- Hello World
- Important features
- Worth mentioning features
- Conclusions

Introduction

- Bjarne Stroustrup, 1979
- Original Goal: C++ = Simula's strengths + C's programming capabilities + Affordability + Portability
- 1998, C++ becomes an International Standard (ISO)
- Used in several application areas:
  - ✓ Operating systems (Windows, Linux, Mac OS)
  - ✓ Popular applications (Visual Studio, MS Office)
  - ✓ Game engines (Unreal Engine, Unity)
  - ✓ Microcontrollers
  - ✓ Teaching & Research

Rules of thumb

- C++ development design principles
  1. General rules – focusing on the C++ community
  2. Design support rules – focusing on data abstraction & OOP concepts
  3. Language-technical rules – focusing on how things are expressed in C++
  4. Low-level programming support rules – influence C++ as a language that express high level designs

Hello World

```
#include <iostream>
// gives the compiler instructions to ”include” in the program the standard input
and output mechanisms
using namespace std;
// use the standard toolbox which includes the tools cout and endl
int main() {
  cout << ”Hello World!” << endl;
  return 0;
}
```

Important features 1/3

- Object Oriented Principles in C++
  - ✓ Not present from the beginning
  - ✓ OOP support was crucial for the spreading of C++
  - ✓ Stroustrup had to publish a journal in 1995 to support that C++ is not only an OOP language
  - ✓ were among the design ideas that he wanted C++ to support

Important features 2/3

- Dynamic Memory Allocation (DMA)
  - ✓ Manage data during compilation
  - ✓ Dynamic allocated memory based on current demands
  - ✓ Influenced by the DM in C
  - ✓ Efficient memory management for different scenarios (e.g. arrays with unknown size)

Important features 3/3

- Standard Library
  - ✓ Collection of components with identical behavior
  - ✓ The most important library
  - ✓ 400 documentation pages > the pages that define the C++ language (300)
  - ✓ Key Criteria for including a class in the library:
    1. Can be used by both beginners and experts
    2. Should be in a generic form
    3. Should be simple to understand and learn
  - ✓ Most important part: Standard Template Library (STL), consists of containers, algorithms and iterators

Worth mentioning features

- Smart Pointers
  - ✓ Save the programmer from possible memory leaks
  - ✓ Automatically manage memory
  - ✓ Deallocate objects when they are not in use
- Modules as an extension
  - ✓ Replace header files and the preprocessor #include directive
  - ✓ Resolve several C++ issues (e.g., header-order dependencies, cyclic dependencies, multiple same header and more)
  - ✓ Improved code structure, faster build times & less error-prone code
- Mixing C & C++ code
  - ✓ Use legacy code or provide specific functionality with a C library
  - ✓ Stroustrup et al created a C++ FAQ blog to describe how can this happen

Conclusions

- C++ is one of the most popular and widespread programming languages
- Initially designed as a C dialect later got extended and became an International Standard
- Today is used in hundreds of thousands of programs
- Several important features:
  - ✓ OOP support
  - ✓ DMA
  - ✓ Standard Library
- Several worth mentioning features:
  - ✓ Smart Pointers
  - ✓ Modules as an extension
  - ✓ Mixing C with C++ code

Thank you for your attention

Object Example

```
class Circle{
        cont double PI = 3.14159;
        private:
        double radius;
        public:
        double diameter;
        Circle (double diameter); //constructor

        double Area(){
                radius = diameter/2.0;
                return PI * radius * radius;
        }
};

int main(){

        Circle circle(6.0);
        //Creates a circle with diameter = 6.0

        circle.radius = 3.0
        //Illegal because radius is private

        cout << circle.Area();
        //Will give as result 28.27

        return 0;
}
```

DMA Example

```
#include <iostream>
using namespace std;

int main() {
    int num;
    cout << "Enter the number of the students: ";
    cin >> num;
    int* ptr;

    // memory allocation of num number of integers
    ptr = new int[num];

    cout << "Enter matriculation number of students." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << ": ";
        cin >> *(ptr + i);
    }

    cout << "\nDisplaying matriculation numbers of students." << endl
        ;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << " :" << *(ptr + i) << endl;
    }

    // ptr memory is released
    delete[] ptr;

    return 0;
}
```

Rules of thumb

General Rules

Real-world issues must guide C++'s evolution.
Do not get involved in a sterile quest for perfection.
C++ must be useful now.
Every feature must be implemented in a way that is fairly clear.
Always provide a transition path.
C++ is not a complete system; it is a programming language.
Provide comprehensive support for each supported.

Design support rules

Support sound design notions.
Provide facilities for program organization.
Say what you mean.
All features must be affordable.
It is more important to allow a useful feature than to prevent every misuse.
Support composition of software from separately developed parts.

Language-technical rules

The static type system should not be violated in any way.
Provide user-defined types with the same level of support as built-in types.
Locality is good.
Avoid other dependencies.
If in doubt, choose the easiest-to-teach variant of a feature.
Syntax matters (often in perverse ways).
Per-processor usage should be eliminated.

Low-level programming support rules

Use traditional (dumb) linkers.
No gratuitous incompatibilities with C.
No lower-level languages below C++ (except assembler).
You don't have to pay for what you don't use (zero-overhead rule).
If in doubt, offer means for manual control.

Mixing Code

http://www.parashift.com/c++-faq-lite/c-calls-cpp.html

If C++ is the most powerful, why isn't it the most popular?

C++ is incredibly powerful because it is very close to the CPU. It makes your code real fast but you have to write loads of code. Doing multi threads or async programming with C++ is nightmare of the worst kind. It is very hard to use. But almost all operating systems in use today are written in C++. Almost all drivers on your computer are written in C++.

Then things changed. CPU's became multi core and today you can have 12 cores on 1 cpu, but each core runs way slower than earlier models. So to get good speed, you must write multi threaded code. This is one reason why C++ is less popular today than it was.

There are just cases where new languages and tools are better to solve your problem in way less time.

https://www.quora.com/If-C++-is-the-most-powerful-why-isnt-it-the-most-popular

COBOL

By Meghana Shamkant Kale,
Concepts of programming languages

AGENDA

- Introduction
- Design Goals
- Structure
- Coding Rules
- Advantages and Limitations
- A Basic Cobol Program
- Conclusions

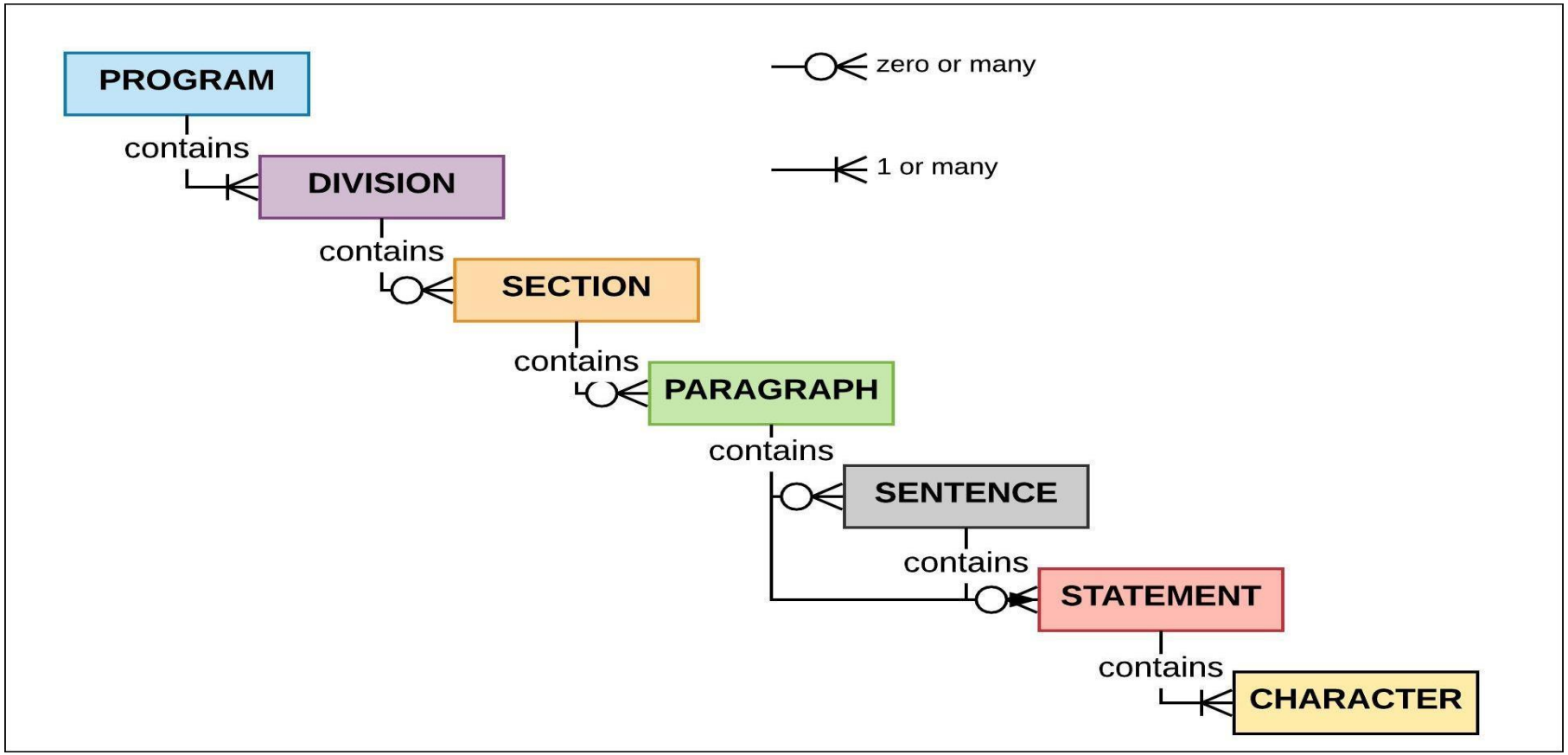
INTRODUCTION

- COBOL stands for “Common Business Oriented Language”.
- Development started in 1959.
- First Compilers developed in 1960.
- Major use in Finance, Government and Military applications.
- Long-lived language.
- Irreplaceable
- 80% of all financial transactions.
- 75% of all the computer transactions

DESIGN GOALS

- Simple English like structure.
- No complex functionalities.
- Machine independent.
- Readable.
- Domain Specific.
- Commonly understandable.

STRUCTURE


PROGRAM
contains
DIVISION
contains
SECTION
contains
PARAGRAPH
contains
SENTENCE
contains
STATEMENT
contains
CHARACTER
zero or many
1 or many

CODING RULES

Columns	Use
1 to 6	Sequence numbers or page and line numbers(optional)
7	Continuation comment or starting a new page
8-11	Area A
12-72	Area B
73-80	Program Identification(optional)

COBOL CODING SHEET

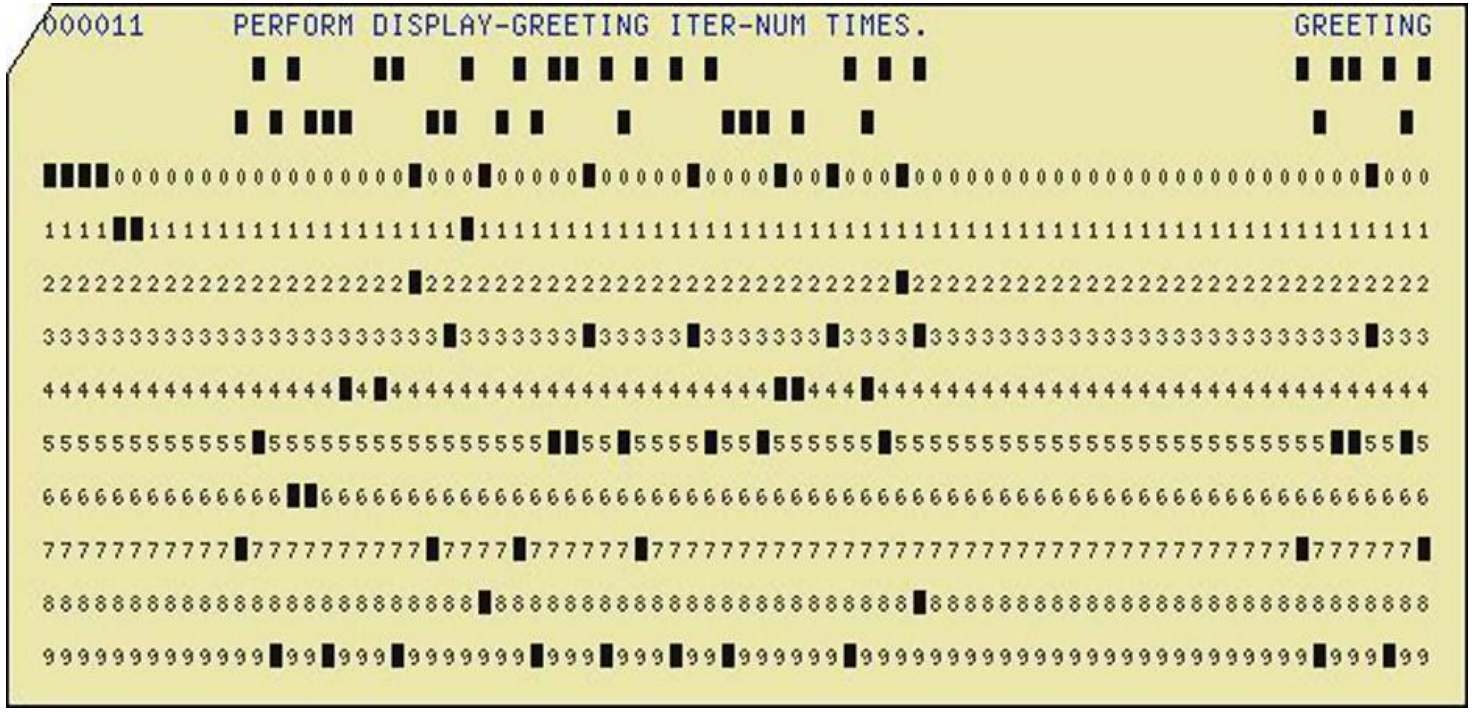
COBOL Coding Sheet

Program: COBOL-GREETING

Programmer: MICHAEL COUGHLAN

Sequence Numbers		Area A	Area B
000001		IDENTIFICATION DIVISION.	
000002		PROGRAM-ID. COBOL-GREETING.	
000003	*	PROGRAM TO DISPLAY COBOL GREETINGS	
000004			
000005		DATA DIVISION.	
000006		WORKING-STORAGE SECTION.	
000007		01 ITER-NUM PIC 9 VALUE 5.	
000008			
000009		PROCEDURE DIVISION.	
000010		BEGIN.	
000011			PERFORM DISPLAY-GREETING ITER-NUM TIMES.
000012			STOP-RUN.
000013			
000014		DISPLAY-GREETING.	
000015			DISPLAY "GREETINGS FROM COBOL".

PUNCH CARD



Adapted from “Beginning CBOL for Programmers”

ADVANTAGES

- Self Documenting
- Stability
- Simple
- Non-proprietary
- Maintainability

LIMITATIONS

- Noise words
- Old Structure
- Long list of key-words
- Wordy programs
- Large dependency in critical applications

BASIC COBOL PROGRAM

```
IDENTIFICATION DIVISION.                                  → DIVISION WITH PROGRAM INFORMATION
PROGRAM-ID. HELLO.                                        → MANDATORY PARAGRAPH
DATA DIVISION.                                            → DIVISION WITH VARIABLE DECLARATION
  WORKING-STORAGE SECTION.                                → DATA DECLARATION SECTION
   01 WS-STUDENT-NAME PIC A(25).                          → ELEMENTARY DATA ITEM
   01 WS-COLLEGE-NAME PIC A(50).                          → ELEMENTARY DATA ITEM
   01 WS-CLG-ID-NUM      PIC 9(03)  VALUE 100.            → ELEMENTARY DATA ITEM
   01 WS-GRADES          PIC  S9(1)VS9(1).                → ELEMENTARY DATA ITEM

   01 WS-EXAM-GRADES                                      → GROUP DATA ITEM
   01 WS-SUB1   PIC S9(1)VS9(1) VALUE 1.5                 → ELEMENTARY DATA ITEM
   01 WS-SUB2   PIC S9(1)VS9(1) VALUE 1.0                 → ELEMENTARY DATA ITEM

PROCEDURE DIVISION.                                       → PROGRAM LOGIC DIVISION
  A000-FIRST-PARA.                                        → FIRST PARAGRAPH
  ACCEPT WS-COLLEGE-NAME.                                 → INPUT AND OUTPUT
  DISPLAY 'STUDENT DATA'.                                 → INPUT AND OUTPUT
  MOVE ‘ABC’ TO WS-STUDENT-NAME.                          → ASSIGNMENT
  DISPLAY “STUDENT NAME IS :”  WS-STUDENT-NAME.           → INPUT AND OUTPUT
  DISPLAY “COLLEGE NAME IS :”  WS-COLLEGE-NAME.           → INPUT AND OUTPUT
  DISPLAY “STUDENT ID :” WS-CLG-ID-NUM.                   → INPUT AND OUTPUT
  COMPUTE WS-GRADES = (WS-SUB1   + WS-SUB2) / 2.          → AIRTHMETIC
  DISPLAY “EXAM RESULT IS :”  WS-GRADES.                  → INPUT AND OUTPUT
STOP RUN.
```

SUMMARY

Attributes	Values
Developers	CODASYL, ANSI, ISO
Released	1959
Stable Versions	4(ANS 64, ANS 74, ANS 85, ISO 2022[Object Oriented])
Typing	Static
Interpreted/Compiled	Compiled
Domain	Large data oriented applications, Business
Paradigm	High-level, Imperative, Procedural and Object-oriented since 2002

CONCLUSIONS

- Cobol is a old, long-lived, high level, domain specific language.
- It is easy to learn and understand.
- It is machine independent.
- Used for many mission critical applications.
- Still in use due to its stability.

Thank You

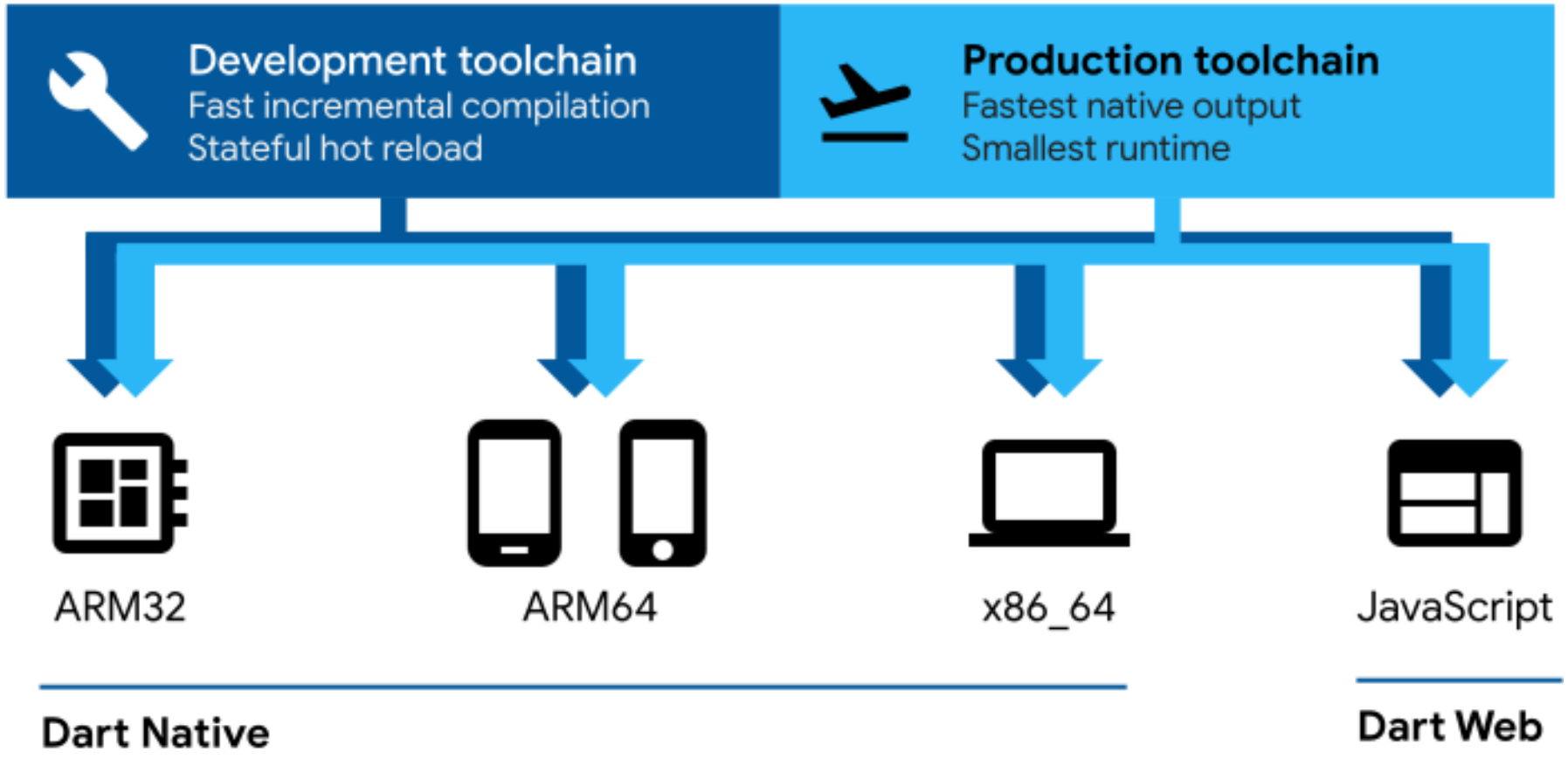
```
int mins = 5 + 2:
print("# Welcome to $mins minutes of **Dart**!");
print("## Philipp Kurrle");
```

Welcome to 7 minutes of Dart!

Philipp Kurrle

Overview

- Designed at Google and released in 2013
- Goal: **Easier** and **faster** development of applications
- Dart can scale from small scripts to large applications
- Designed to be **easy to learn** for developers with knowledge in Java, C# etc.
- **Class-based**, **object oriented** and **dynamic**
- Compiles to JavaScript


Development toolchain
Fast incremental compilation
Stateful hot reload
Production toolchain
Fastest native output
Smallest runtime
ARM32
ARM64
x86_64
JavaScript
Dart Native
Dart Web

Design Principles

1. Everything is an object

- Every value at runtime is an object.
  - Even elementary data such as strings and booleans

Design Principles

1. Everything is an object

- Every value at runtime is an object.
  - Even elementary data such as strings and booleans

2. Program to an interface, not an implementation.

- Types are based on interfaces, not on classes.
- All acces to state is done via accessor methods
- No final methods.

Design Principles

3. Optional Typing

- Type identifiers are optional
- Safe and sound type system

```
var name = "Bob";
// is the same like
String name = "Bob";
```

Basic Program Structure

```
// Define a function.
void printInteger(int aNumber) {
    print('The number is $aNumber.');
}

// This is where the app starts executing.
void main() {
  var number = 42;
  printInteger(number);
}
```

Basic Program Structure

```
// Copyright 2018 The Flutter team. All rights reserved.

import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Evaluation

Readability

- **Simple**, recognizable for most developers
- Optional typing might decrease readability
- No **feature mulitplicity**
- Orthogonal
    - Few Features that can be used in many ways
- Sufficient number of **data types** and **control statements**

Evaluation

Writeability

- Supports **abstraction**
  - Few features that can be combined in many ways.
- Dart is **expressive**
  - Specification can be translated into the language without "bending it"

Evaluation

Reliability

- Dart uses **Type-Checking**
- Exception Handling

```
try {
    // code
} catch(e) {
    // catch code here
}
```

- Aliasing is possible

Evaluation

Cost

- Easy to learn language
- Free IDEs
- Programs can be developed fast (especially with Flutter)

Cheatsheet

Question	Answer
Object oriented	yes
Macro language	no
Typing?	optinal
Use case?	mobile apps, web applications
Release?	2013
Interpredted/Compiled?	compiled to JavaScript
Level?	High level
Number of keywords?	average number of keywords
Type interfenece?	type annotations are optional because of type interference

Sources

- Dart platforms: https://dart.dev/overview
- Code picture: http://www.newsmaritime.com/wp-content/uploads/2016/06/Check-out-the-new-and-functional-programming-language.jpg
- iPhone https://flutter.dev/docs/get-started/codelab
- Dart Logo https://dart.dev/

Thank you for your attention!

FORTH

Concepts of Programming Language SS21

Sayali Kakade

(02kasa1mst@hft-stuttgart.de)

Agenda

- Introduction
- Basic Components of Forth
- Error and Exception Handling
- Forth in the Real World
- Summary
- References

Introduction

- Charles (Chuck) Moore developed FORTH in early 1970s
- First stand-alone FORTH implementation was for the National Radio Astronomy Observatory radio telescope in Arizona
- Forth is interactive and extensible
- Meta-application language
- Uses Reverse Polish(Postfix) Notation (for addition – type 3 5 + instead of 3+5)
- Compiled code is small and portable
- Fundamental program unit, “words”, similar to subroutines or commands in other languages
- Can be implemented on any mini- or micro-computer

Basic Components of Forth

- **Dictionary**
  - The ”dictionary” contains all system-defined or user-defined words
  - Self-defined words are defined using “:” and end with “;”
  - Example- : LIFT HAND OPEN ARM LOWER HAND CLOSE ARM RAISE ; [3]
- **Data Structures**
  - The body of data structure commands comprises the real data.
  - CONSTANT is used in the following context: 10 CONSTANT TEN
  - The word VARIABLE allocates a cell and compiles a word which pushes the address of that cell on the stack.
  - The data at a location can be stored by ! and fetched by @

Basic Components of Forth

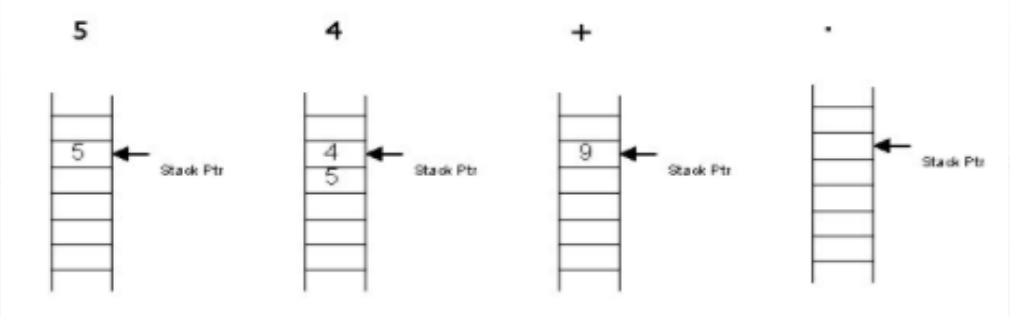
- **The Stack**
  - works on a last-in, first-out (LIFO) principle.
  - Postfix notation in which operands come before operators.

    Data stack:
    - holds the arguments passed between the words and return values

    Return stack:
    - to store return addresses for nested definitions and loop control parameters
    - can be used for temporary data storage
  - For example-

    > 5 4 + . 9 ok

  To take a look at the stack use the .s command



Basic Components of Forth

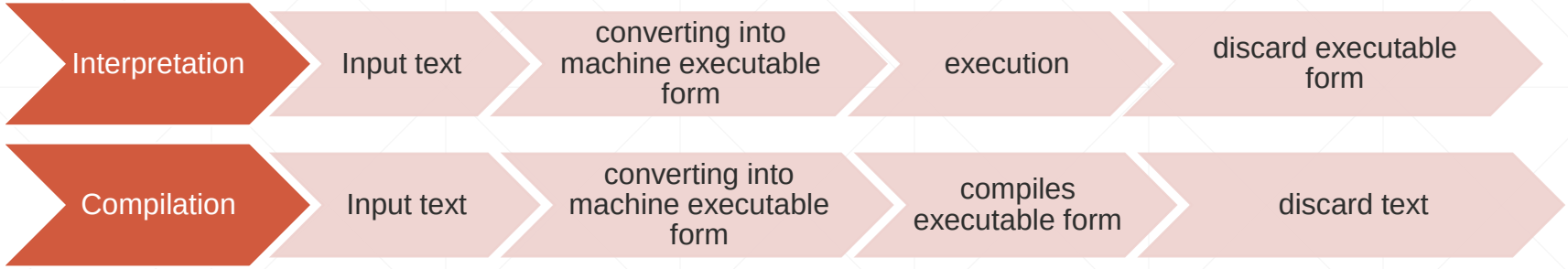
- **Stack Operations**

  As this is a stack machine, there a few operations that focus on manipulating the stack

  - DUP – duplicates the top of the stack
  - SWAP – swaps the top 2 elements
  - DROP – drops the top element of stack
  - ROT – Rotate the top 3 numbers

Basic Components of Forth

- **Interpretation and Compilation state**



For instance - : squared (n1 -- n1^2) dup * ;

- : tells compiler to define a new word ‘squared’ and compile into the word the actions that follow
- ; stops the compiler from compiling and returns it to the mode of executing the addresses instead

Basic Components of Forth

- **Immediate words**
  - The word IMMEDIATE replaces the compilation semantics of the most recent colon definition with the interpretation semantics.
  - Example - the ; (semi-colon) is an immediate word
- **POSTPONE**
  - The word POSTPONE takes a name as a parameter and appends the compilation semantics of the specified word to the current definition.
  - POSTPONE facilitates use of words such as IF and THEN inside other colon definition.

Basic Components of Forth

- **Control Structures**

  In Forth, control structures allow for execution and looping based on stack values.

  - IF ... THEN
  - IF...ELSE...THEN
  - EXIT
  - DO ... LOOP
  - BEGIN ... AGAIN
  - BEGIN ... UNTIL
  - BEGIN ... WHILE ... REPEAT
  - CASE ... OF ... ENDOF ... ENDCASE

Error and Exception Handling

- **ABORT, QUIT and ABORT”:**
  - ABORT resets the data stack and runs Forth text interpreter by calling QUIT
  - QUIT resets the return stack, forces input from system console
  - ABORT” (x “<text>” -) , if x is non-zero resets the data stack, usually displays the text as a message, and then runs QUIT
- **CATCH and THROW:**
  - CATCH is similar to EXECUTE when it returns 0 (no THROW executed)
  - When CATCH returns a non-zero value (THROW executed), the depth of the stack, not counting the throw value, is the same as it was just before CATCH’s argument word was executed
  - It is only the depth of the stack that is preserved, not the contents of the stack

Forth in the Real World

Few of the Forth implementations in the real world can be seen in areas such as -

- Process Control (monitor robot motion-picture cameras in movies)
- Portable intelligent devices (an automotive ignition analyser)
- Medical domain (a heart monitor)
- Image processing (NASA's Goddard Space Flight Center)
- Data acquisition and analysis

Summary

- Forth is a stack-based, extensible programming language without type-checking.
- Programming consists of defining new words in terms of existing ones.
- Since the entire language structure is embodied in words, the application programmer can "extend" Forth to add new operators, program constructs, or data types at will.
- Forth runtime, including compiler, editor, “file system”, “virtual memory” can be implemented in a few kilobytes of memory.
- Incremental development, a fast program-debug cycle, full interactive access to any level of the program, and the ability to work at a high "level of abstraction"

References

[1] Rather, Elizabeth D.,Colburn, Donald R., Moore, Charles H., The Evolution of Forth, Association for Computing Machinery, New York, NY, USA, 1996. https://doi.org/10.1145/234286.1057832

[2] Brodie, Leo, Starting FORTH, isbn: B00072QXAW, Forth Inc.,1980.

[3] Stephen Pelc, Programming Forth, MicroProcessor Engineering Limited, Revised version May 2011.

[4] Dr.Tim Hendtlass, Real time Forth, F-PC Version 5.0, November 1993.

[5] Brodie, Leo, 2004, Thinking Forth, Punchy Publishing.

[6] Gforth reference manual (version 0.7.0, November 2, 2008).

[7] https://www.complang.tuwien.ac.at/forth/faq/faq-general-3.html

[8] http://www.elilabs.com/docs/forth/forth.html

Thank you for your attention!

OVERVIEW OF SMALLTALK

Santhosh kiran Selvam,Summer-2021

AGENDA

- Introduction
- Smalltalk History
- Smalltalk Terminologies
- Error Handling
- Sample Program
- Reference

INTRODUCTION

- Smalltalk was developed in the 1970s for educational purpose.
- Smalltalk consists of its own user interface and an entire development environment.
- Smalltalk consists of a few new programming concepts to acknowledge in order to understand Smalltalk.
- In Smalltalk, objects and messages play an important role and are to implement the complete programming environment.

SMALLTALK HISTORY

- Smalltalk was an object-oriented programming language developed by the Learning Research Group(LRG), led by Alan Kay at Xerox PARC. Alan Kay's vision for Smalltalk was that it should be easy to use by children's and they were able to program their systems.
- Four business partners were awarded licenses to build Smalltalk for their own systems: Tektronix, DEC, Apple, and HP. Smalltalk is one of the most prominent and most successful programming languages of the 1970s.
- Steve Jobs was impressed by Smalltalk's ability to make the Lisa and Macintosh computers.

SMALLTALK TERMINOLOGIES

- Object and Messages
- Class and Instances
- Sample Hello World
- Object Deputy Model
- Array in Smalltalk
- Sector and Argument
- Abstract class

SMALLTALK TERMINOLOGIES

Object and Messages

- Objects represent data structures and are used to store and retrieve information. Objects represent locations and places where any activity necessitates investigation depending on their contact with them.

- A message is sent to an object, instructing it to execute one of its operations. It only addresses the procedure that must be started, but it makes no note of how the operation must be carried out

Example: a foo: b bar + c baz;

Classes and Instances

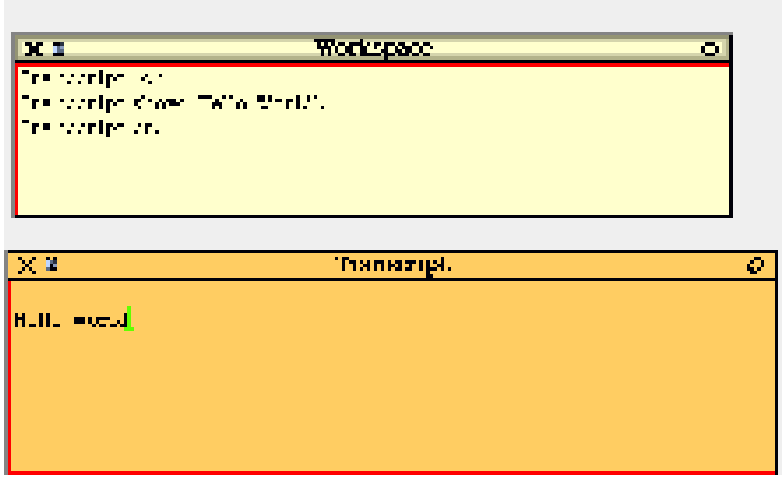
- A class is defined as the application of a collection of objects that all depict a similar type of system constituent.
- The instances are the entity of the object described by a class. A class illustrates the type of its instances' memories that are private and it illustrates how they perform their operations.

Syntax:

```
someClass
subClass: #'nameOfNewClass'
instanceVariableNames: '...list of private slots...'
classVariableNames: '...list of class variables...'
poolDictionaries: '..list of imported pool bindings...'
category: 'for documentation only'.
```

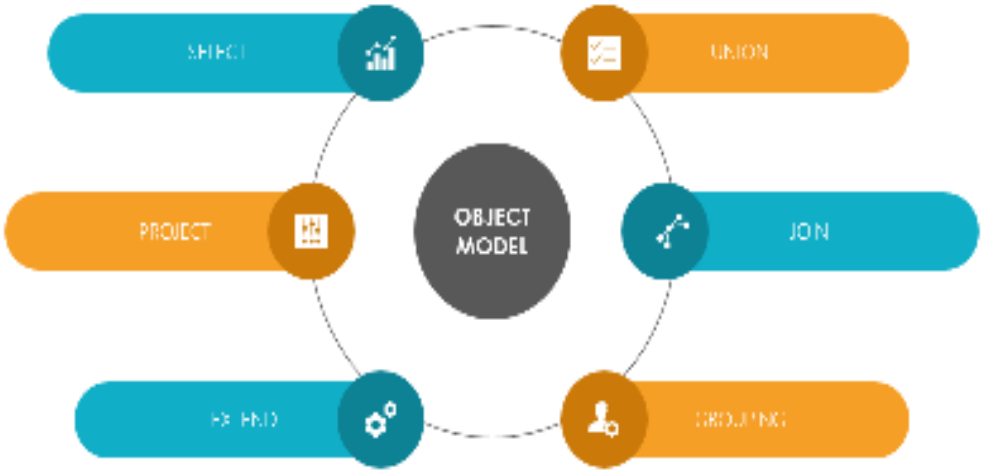
Smalltalk Hello world

Transcript show: 'Hello world!'



Object Deputy Model

- There was a need to prolong the object-oriented model. This could be done by deputy objects and deputy classes.
- The name deputy objects describe the customization of objects and prolonging them. The deputy objects are determined by deputy classes that are deduced by generating deputy objects.



Arrays in Smalltalk

- An array is defined as a simple structure that contains data, and whose details can be referred by an index of an integer.
- The literals such as symbols, characters, strings, and arrays –bounded by round parentheses and preceded by a hash sign.

Example:

#(1 2 3 4 5)

#(‘arts’ ‘science’ ‘commerce’ ‘mathematics’ ‘history’)

Pseudo Variables

NIL

Variables that are not initialized point to nil

TRUE

Is employed to give a positive response to a message that is making a plain yes-no inquiry.

FALSE

Is employed to give a non-affirmative response to a message that is making a plain yes-no inquiry

Selector and Arguments

A message equation consists of selectors and arguments along with a receiver. The selector is mentioned literally, whereas, arguments and receivers are depicted by different expressions.

Example : Theta cos

Abstract classes

Abstract superclasses are generated when two classes have a part of their descriptions in common and still, neither one is perfectly a subclass of the other class.

ERROR HANDELING

Smalltalk used an instance based exception handling scheme, where instances of Signal where created and raised.

SYNTAX: [try block to be evaluated] on:exception do:[:ex | handler code]

Handler Option	Function
ex return	return out of the try block
ex restart	restart the try block
ex reject	handler cannot handle; rethrow the exception for an outer handler
ex proceedWith: value	proceed after where the exception was raised (after a repair)

SAMPLE PROGRAM

```
Object class: #Vehicle
    instanceVariableNames: 'speed wheels'
   classVariableNames: ' '
    PoolDictionaries: ' '
    category: ' '

       withWeels: numberOfWheels goingSpeed: aSpeed
           "Creates a new Vehicle Object"
            | aNewVehicle | aNewVehicle := self new.
           aNewVehicle wheels := numberOfWheels.
            aNewVehicle speed := aSpeed.
           ^aNewVehicle.
      driveOn: aRoad
           "Returns the reciever, does the driving"
           self speed < aRoad speedLimit
            ifTrue:
                [self speed := (self speed) + 1. ^self]
            ifFalse:
               [self speed := (self speed) - 1. ^self].
```

REFERENCE

[1] Goldberg, Adele, and David Robson. Smalltalk-80: the language and its implementation. Addison-Wesley Longman Publishing Co., Inc., 1983.

[2] Ingalls, Daniel. The evolution of Smalltalk: from Smalltalk-72 through Squeak. Proceedings of the ACM on Programming Languages 4.HOPL (2020): 1-101.

[3] Peng, Zhiyong, Kai Huang, and Juan Ye. Development of smalltalk in China. First Conference on Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings.. IEEE, 2003.

Thank You

24Slides

Go Programming Language

By

Vaidehi Dabir

Concept of Programming Language

Summer 2021

Agenda

- Introduction
- Why Golang was created?
- Features
- Who use Go?
- Conclusion

Introduction

- Created at : Google
- Created by : Robert Griesemer, Rob Pike and Ken Thompson
- “C for 21st century”
- ‘Simplicity’ was the key goal

Why Go is created?

- Only three choices left for choosing the language, which were not available in a single language :

  1. efficient compilation
  2. efficient execution
  3. ease of programming

  Hence, Go was born out of frustration.

FEATURES

Simplicity

- ‘:=‘ for declaration as well as initialization of the variable

  Example : var:=6

- No terminator

  Example :

```
package main
import ” fmt ”
func main ( ) {
fmt . P r i n t l n ( ” Hel lo , World ! ” )
}
```

Concurrency and GoRoutine

- “Managing or dealing with multiple tasks at the same time is called as Concurrency.”
- Built-in functionalities and library support concurrency
- Goroutine : functions capable of running concurrently with other functions
- Goroutine Communication

Error Handling in Go

Defer-Panic-Recover :

- Defer : For clean-up activities
- Panic : Stops regular flow of execution and starts panicking
- Recover : Returns nil in case of normal execution, else return the stored values.

Who use Go?

- Netflix
- Adobe
- Uber
- Docker
- Apples
- Dropbox
- Nokia
- Soundcloud
- Google
- Heroku

Conclusion

- Go is a simple, open source, statically-typed, compiled, readable language
- Go provides ability to write concurrent programmes and ease of programming
- Go is used in multiple high-end applications
- Its demand is increasing because of its simplicity, capability to handle concurrency.

Thank You!

GROOVY

KESHAV DESHPANDE

CONCEPTS OF PROGRAMMING LANGUAGE

AGENDA

- INTRODUCTION
- LANGUAGE VISION
- INTERESTING ASPECTS OF THE LANGUAGE
- GROOVYC AND GROOVYCONSOLE
- CRITICISM AND CHALLENGES
- CONCLUSION

INTRODUCTION

- Designed by James Strachan and Groovy 1.0 was released on January 2 2007
- Object oriented programming language
- Main goal was to keep the language dynamic, but later static features were added
- Code gets converted to bytecode

LANGUAGE VISION

- Better for script writers
- Better for java developers
- Better for data scientists/ DSL
- Has the vision changed?

INTERESTING ASPECTS OF LANGUAGE

- Command chains
  - method_1 args_1 [method_n args_n]*
  - take(coffee).with(sugar, milk).and(liquor) gets converted to
  - take coffee with sugar, milk and liquor
- Closures
  - A closure in Groovy is an open, anonymous, block of code that can take arguments, return a value and be assigned to a variable
  - { [closure parameters ->] closure body}

INTERESTING ASPECTS OF LANGUAGE

- Operator overloading

```
c l a s s B uc ket {
  i n t s i z e
 B uc ket ( i n t s i z e ) {
  t h i s . s i z e = s i z e }
 B uc ket p l u s ( B uc ket o t h e r ) {
  r e t u r n new B uc ket ( t h i s . s i z e + o t h e r . s i z e ) }
}
def b1 = new Bucket(4)
def b2 = new Bucket(11)
assert (b1 + b2).size == 15
```

GROOVYC AND GROOVYCONSOLE

- groovyc MyClass.groovy
- groovyConsole - The Groovy swing console

CRITICISM AND CHALLENGES

- Performance issue in some cases
- Integration with android

CONCLUSION

- Easy for creating DSL's
- Acts as a glue between scripting and programming language
- Combination of dynamic and static typed language
- Provides functional aspects through closures
- Is object oriented as it is based on Java

THANK YOU FOR THE ATTENTION

- QUESTIONS?

Code for closure

```
def myConst = 5
define myConst = { num -> num + myConst }
println incByConst(10) // => 15
```

Kotlin

Kai Widmaier

Agenda

- Introduction
- History
- Concepts & Features
- Conclusion

Introduction

- Object oriented, JVM-based language
- Kotlin improves on the weaknesses of Java
- Removes a lot of boilerplate code

```kotlin
data class ProgrammingLanguage(var name: String, var version: String)
```

==

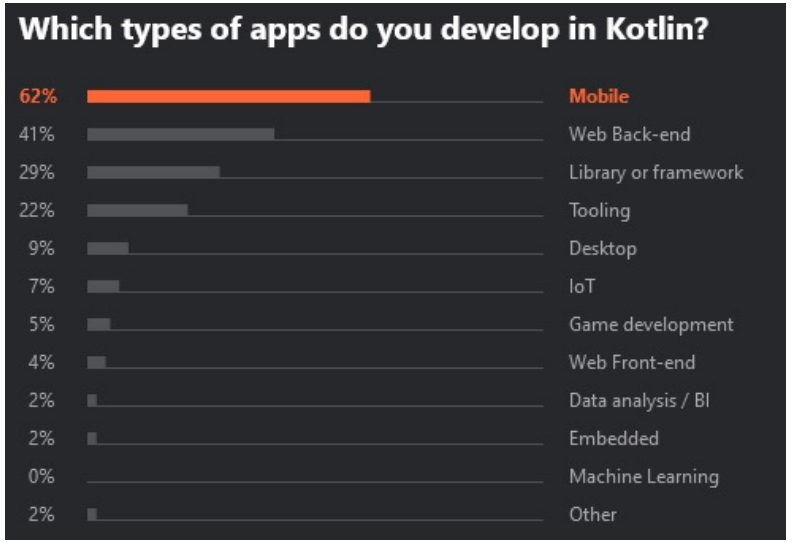
```java
public class ProgrammingLanguage {
  private String name;
  private String version;
  public ProgrammingLanguage(String name,
                             String version){
    this.name = name;
    this.version = version;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getVersion() {
    return version;
  }
  public void setVersion(String version) {
    this.version = version;
  }
}
```

History

- 2010: Development started by JetBrains
  - Goal: JVM-based language that allows programmers to write „concise and crisp code to save time“
- 2016: First stable version released
- 2017: Google announces official support of Kotlin for Android development
- 2018: Kotlin is now the official language for Android development

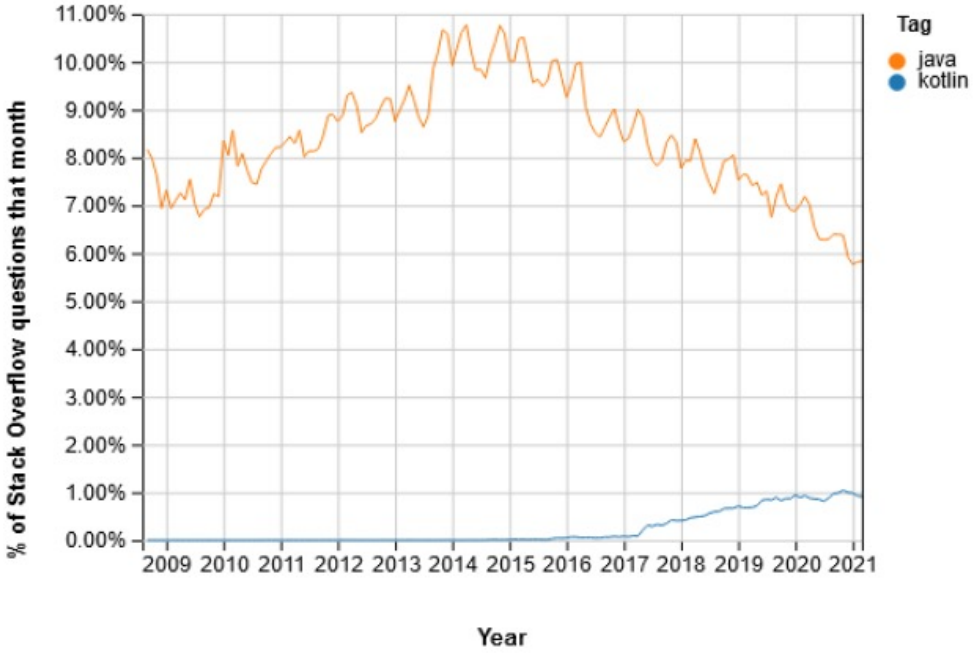
History

2019:



Source: https://www.jetbrains.com/lp/devecosystem-2019/kotlin/

Today:



Source: https://insights.stackoverflow.com/trends

Concepts & Features – Java Interoperability

Kotlin allows to use Java code within Kotlin files since both run on the JVM and use Java Bytecode

Advantages:

- Developers can easily transition to Kotlin
- Java projects can be gradually converted
- Existing libraries and frameworks for Java can be used for Kotlin

Concepts & Features – Null safety

- NullPointerExceptions are (almost) impossible in Kotlin
- Kotlin won‘t let you compile code that could result in a NPE

```
fun printNullReference(){
  val nullReference : String? = null
  println(nullReference.toUpperCase())
}
```

- Null checks have to be done with things like if-statements

Concepts & Features - Exceptions

Kotlin only uses unchecked exceptions

Advantages:

- Code is easier to read because there are no unnecessary try/catch blocks where they are not needed
- Checked exceptions don‘t lead to more productivity / increased code quality

Concepts & Features - Extensions

Kotlin uses extensions to add functionality to classes
Extension functions are functions that can be added to classes from anywhere in the code

```
fun <T> ArrayList<T>.getFirst(): T?{
  return if(this.size > 0){
    this[0]
  }
  else{
    null //Or throw some exception
  }
}
```

```
fun someFunction(){
  val arrLst = ArrayList<Int>()
  arrLst.getFirst()
}
```

The same thing can be done with properties!

Concepts & Features - Extensions

Kotlin doesn‘t use the „static“ keyword.

Alternative: Object extensions

- Companion objects can be added to any class
- Can be used in the same context as static properties and methods in Java

```
class ExtensionExample{
  companion object CompanionObject{
    var value = 5
  }
}
class SomeClass{
  fun someFunction(){
    println(ExtensionExample.value)
  }
}
```

Conclusion

- Kotlin allows for a more efficient way of writing code
- Future seems promising because more and more people start using Kotlin
- Enterprise environment tend to be conservative with choices programming languages and Java will probably still be popular in this field for a long time

CoPL – Lua

AN EXTENSIBLE EXTENSION LANGUAGE

Agenda

1. History
2. Lua – Properties
3. Data types
4. C API
5. Secret niche of Lua
6. Typical usage
7. Reference list

Figure 1 – Lua logo

Copyright © 1998 Lua.org. Graphic design by Alexandre Nakonechnyj.

History

- Lua – Created 1993 in Rio de Janeiro
  - Designed by Roberto Ierusalimschy, Waldemar Celes, Luiz Henrique de Figueiredo
- PETROBAS (Brazilian Oil Company)
  - Two projects
    - Simulation several times a day (legacy code; strict)
    - PGM (configurable report generator for lithology profiles)
- DEL (data entry language)
- Sol (Simple Object Language)
- Lua = "DEL".." + ".."Sol" print(Lua) → DEL + Sol

Lua – Properties

- provide data-description facilities
- user-friendly (avoid cryptic syntax)
- portable
- runnable on many machines (PC DOS, Unix)
  - And gaming consoles → LuaJIT

Figure 2 – LuaJIT compatibility

Copyright © 2005-2021 Mike Pall

Data Types

- boolean
- string
- number (> 100,000,000,000,000 → rounding errors; Lua 5.3)
- userdata (generic C pointer)
- function (Lua function)
- thread (coroutine)
- table (arrays)
- nil

```
#include <stdio.h>
#include <lua.h> // Lua main library (lua_*)
#include <lauxlib.h> // Lua auxiliary library (luaL_*)
```

C API

- Lua → not a stand-alone language (but possible)
- Lua acts as library (extension)
- Host language → Non-native Lua code → Lua is extensible

- Minimalistic as Lua
- Drawbacks (multiple calls)
- Stack-based (push; pop C data types)
  - Differs from traditional stack (direct index)

C API – A Code Example

```
#include <stdio.h>
#include <lua.h> // (lua_*)
#include <lauxlib.h> // (luaL_*)
int main(void)
    // create a Lua state
    lua_State *L = luaL_newstate();

    // load & execute Lua code
    char a[] = "function foo (x,y) return x+y end"
    if (luaL_dostring(L,a)) {
        lua_close(L);
        return -1;
    }
    lua_getglobal(L, "foo");
    lua_pushinteger(L,5);
    lua_pushinteger(L,3);
    lua_call(L,2,1); //call foo; 2 args, 1 return
    printf("Result: %d\n", lua_tointeger(L, -1));
    // value on top of stack with "-1" (5+3=8)
    lua_pop(L,1); // original stack state
    lua_close(L); // close Lua state
    return 0;
}
```

Secret niche of Lua

- *"Fast, portable, embeddable, powerful, small and free"*
- *Support for*
  - *Procedural, OO, functional, data-driven programming*
  - *Data description*
- *Ranges many applications (SW, TV-Box, game industry)*
- *Used in games and game engines*

Typical Usage

- Non-game related applications
  - VLC, Apache HTTP Server, Vim, MediaWiki, Nmap, Waze, bootloader of FreeBSD
- Programmers perspective
  - Leverage C API („better to have than not to have“)
  - In case of:

Figure 3 – No src code error

Karlsen L. 2008, stack overflow

Reference List

1. *Figure 1. Nakonechnyj, A. 1998. Copyright © 1998 Lua.org. http://lua-users.org/wiki/LuaLogo.*
2. *Figure 2. Pall, M. 2021. Copyright © 2005-2021. https://luajit.org/luajit.html.*
3. *Figure 3. Karlsen, L. 2008. stack overflow. Accessed 24 May 2021. https://stackoverflow.com/questions/314329/getting-rid-of-there-is-no-source-code-available-for-the-current-location.*

print(„THANK YOU FOR YOUR ATTENTION!“) ☺

Concepts of Programming Languages

PROLOG

Kevin Kammleiter

Agenda

- Overview
- Idea of Prolog
- Rules and Facts
- Execution
- Conclusion

Introduction

- Developed in France 1972
- Used in natural language processing, AI
- Declarative logical programming language

Idea of Prolog

- Horn clauses:
  - $z \leftarrow (x \wedge y)$ (implication)
  - assume that x and y hold, then also z holds
- Rules and facts as domain
- Queries on these relations

Rules and Facts

- $Head :- Body.$
  - Head is true if Body is true.
  - Body any number of statements conjunct(,) or disjunct(;)

- $person(john).$
  - Clause without Body
  - states “john is a person”.

- $livingThing(X) :- person(X).$

- Loops can be implemented with recursion

Execution

- $person(john).$ <- fact in database
- $livingThing(X) :- person(X).$ <- rule in database

- Query: >person(john). <- returns true

  >person(X). <- returns X=john

  >livingThing(X). <- returns X=john

Conclusion

- Straight forward logical language
- Query-Proofing
- Widley used in research and education
- Niche language
- Can achieve a lot, but should only be used in its intended field

Thank You

Hochschule
für Technik
Stuttgart

Swift

Concepts of programming languages

Kenza Mekdam

Agenda

01 Introduction and History

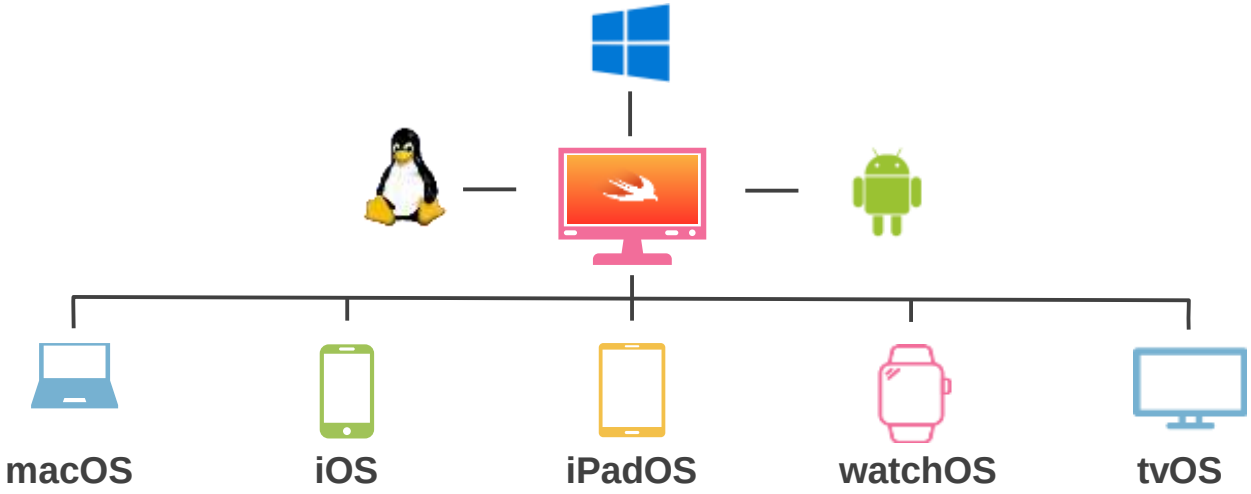
02 Basic Programming

03 Characteristics of Swift

04 Advantages and disadvantages of Swift

05 Conclusion

Swift Introduction

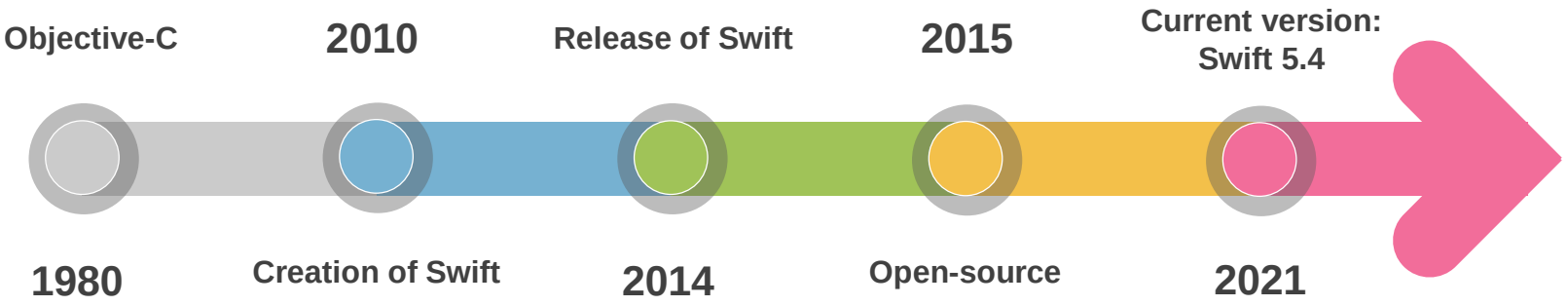


It is a general-purpose, multi-paradigm, compiled programming language.

Developed by Apple Inc. to replace Objective-C and released in 2014.

Friendly to new programmers, it introduces shorter code, and safer programs.

Swift’s History



Swift’s Development Tools

XCode

Coderunner

JetBrains AppCode

Sublime Text

Basic Programming

01

Hello world!

```
print("Hello World!")
```

02

Declaring a variable:

```
var age = 22
```

03

Declaring a constant:

```
let name = Jane
```

04

If-Else statement

```
let temperature = 100
if temperature >= 100 {
print("The water is boiling.") }
else {
print("The water is not boiling."
)}
```

05

Type annotation:

```
var varName: string
```

06

Function

```
func congratulate(person: String) -
>
String{
let congrats = "Congratulations!" +
person + ":)"
return congrats}
print (congratulate(person: "Jane")
)
```

07

Switch statement

```
let character = "z"
switch character {
case "a", "e", "i", "o", "u" :
print("This character is a vowel.")
default:
print("This character is not a vowel."
)}
```

Characteristics of Swift

Similarities with C

Swift is a C-family programming language

Most C operators are supported in Swift, Statements are grouped inside curly braces… etc.

Types

There are two sets of types in Swift:

Named types: Classes, structures, enumerations, & protocols.
Compound types: Function types, tuple types.

Collection types

There are three collection types provided by Swift:

Arrays, sets, & dictionaries.

Type safety and type inference

Type safety:

When compiling the code, it executes type checks, and flags all the mismatched types as errors.

Type inference:

When compiling the code, it allows the compiler to automatically determine the type of a given expression based on the values supplied by the programmer.

Characteristics of Swift

Expressions

Swift provides four types of expressions:

Prefix expressions, Binary expressions, Primary expressions, & Postfix expressions.

Statements

Swift provides three types of statements:

Simple statements, Compiler control statements, & Control flow statements.

Functions

In Swift, any function has a type:

It is made up of the function's parameter types and return type.

Access control

Swift offers five separate access levels for code entities:

Private access, File-private access, Internal access, Public access, Open access.

Characteristics of Swift

Memory safety

Swift prohibits unsafe behavior from occurring in the code by default:

For instance, Variables are initialized until they're used, memory is not accessed after being deallocated, and array indices are tested for out-of-bounds errors.

Automatic reference counting

Swift tracks and manages the app's memory use with (ARC)

This ensures that memory control in Swift works automatically and the programmer doesn't have to worry about it.

Extensions

Extensions give an existing class, structure, enumeration, or protocol type new features

This provides the freedom to expand types for which the original source code is unavailable.

Error handling

Swift has mainly 4 ways of error handling:

- Propagating the error from a function to the code that calls that function.
- Using a do-catch statement.
- Handling the error as an optional value.
- Asserting that the error will not occur.

Comparison of Speed

Language	Seconds	Memory
Swift	5,50	192,172
Java	8,39	933,808
C++	2,62	130,212
C#	8,26	870,360

Test binary-trees description [1],[2].

Advantages and Disadvantages

- ✓ Uses a modern and easy syntax.
  - ➢ This facilitates code design and control.
- ✓ Includes tuples, optionals, closures, and generics.
- ✓ It is open source.
- ✓ Not only for iOS.
  - ➢ But it supports Android, Windows, and Linux as well.

- ○ Continuous changes and accelerated development.
- ○ Smaller extensions.
- ○ Less libraries and frameworks than in other languages.
- ○ Version to version incompatibilities.

Conclusion

Readability

- It has a simple syntax
- It has control statements.
- It has data Types.

Writability

- It is expressive.
- It is readable.

Reliability

- It check types.
- It handles exceptions.

Cost

- Open-source programming language.
- Open-source tools.

References

[1] Rostislav Fojtik. Swift a New Programming Language for Development and Education. Digital Science, 284-295, 2019.

[2] The Computer Language Benchamrks Game, https://benchmarksgameteam.pages.debian.net/benchmarksgame/

[3] Apple Inc. The swift programming language - https://docs.swift.org/swiftbook/LanguageGuide/TheBasics.html

[4] Wikipedia contributors. ”Swift (programming language).” Wikipedia, The Free Encyclopedia.

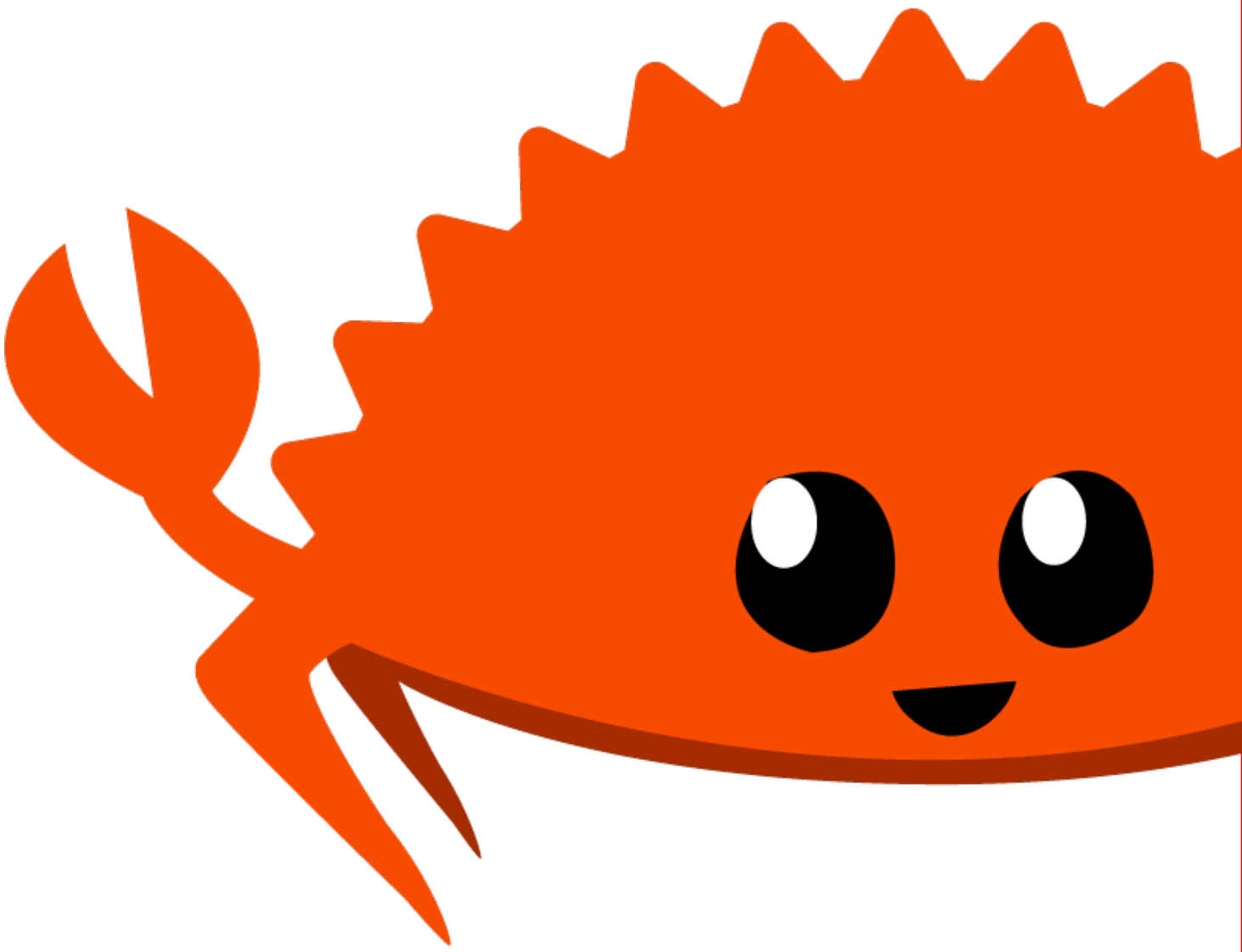
Hochschule für Technik Stuttgart

Thank you for your attention!

Rust

CPL Summer 2021

Brian Platzer



Innovative Hochschule
Eine gemeinsame Initiative von Bund und Ländern

Agenda

- History
- Purpose & Goals
- Basic Characteristics
- When to Use
- Basic Structure
- Dependencies
- Differentiating Factors
- References

History

- Development began in 2006 by Graydon Hoare
  - Researcher at Mozilla Resarch
  - Project initially sponsored by Mozilla
  - Later sponsored by Rust Foundation: Amazon, Google, etc…
- First major release in 2010
- Inspired by C++, ML, Haskell, Swift, C#, Ruby
- **Major Editions:** Rust 2015, 2018, Rust 2021?

Purpose and Goals

- Provide a comfortable level of abstraction improving upon four common pain points of other languages
  - Not enough memory safety
  - Poor support for concurrency
  - Limited control over resources
  - Lack of practicality

Non-Goals

- Not employing cutting-edge technology
  - Because established techniques are better
- Not covering the complete feature set of other languages
  - Because Rust needs to be able to handle just the majority of use cases
- Not insisting that Rust run on all platforms
  - Because Rust should run on simply the majority of hardware/software

Basic Characteristics

- Multi-paradigm
  - OO, Functional, Imperative Procedural
- Very safe
- Statically typed
- High performing
- Compiled
- Handles concurrency well
- Syntactically similar to C++

When to Use

- Weigh criteria of readability, writability, reliability, generality, portability, and costs

- **Readability/Writability:** A C++ developer would be comfortable learning Rust, but it might take longer to write code
- **Reliability:** Rust is very reliable, but one should pay close attention to the dependencies ecosystem for stale packages
- **Generality:** Hundreds of use cases, shines around network services, command-line apps, WebAssembly
- **Cost:** The opportunity cost and actual cost of learning Rust/hiring Rust-capable developers could be the weakest point

Basic Structure

Run `cargo new <name>` and a structure is built as follows

```
<name>
- Cargo.toml                # Dependency declaration file
- src                       # Folder for the source code
      - main.rs
```

Hello World

```
fn main() {
      println!("Hello, World!");
}
```

Dependency Ecosystem

- Native system is called *Cargo*
- Libraries in Rust are called *crates*
- Over 60,000 crates available

```
[package]
name = "Foo_Bar_Hello_World"
version = "0.0.1"
authors = ["Foo Bar <foo@bar.com>"]
[dependencies]
rand = { git = "http://git.com/bar.git" }
```

Differentiator: Variable Mutability

- By default, variables are **immutable**
- You must explicitly tell Rust that your variable is allowed to change, or else you get an error: `cannot assign twice to immutable variable`

```
fn main() {
   let mut x = 1000;
   println!("Value is: {}", x);
   x = 500;
   println!("Value is: {}", x);
}
```

```
The value of x is: 1000
The value of x is: 500
```

Differentiator: Ownership

- Helps with memory safety
- Replaces garbage collection or explicit memory allocation
- Some basic rules exist
  - Every value in Rust can be identified with a variable called its owner
  - Only one owner is allowed at a time
  - When owners go outside of their scope, the value is dropped

- Not enough time to provide an example here, but you can read my paper and find it in section 3.1.2 ☺

Differentiator: Community

- Voted StackOverflow‘s „Most Loved“ language four years in a row – 2021, 2020, 2019, 2018
- Contributors on GitHub
  - 51 in 2012
  - 2000 in 2017
  - 5000 in 2021
- Over 60 companies made it publicly known that they use Rust

Useful References/Links

- Full list of references
  - Please see my paper
- Rust Book
  - http://doc.rust-lang.org/book
- Rust Playpen
  - https://play.rust.org

Mini Cheatsheet	
Paradigms	Object oriented, imperative, functional
Macro language	Yes
Typing	Static, must be known at compile time
Use Cases	Network services, command line apps, WebAssembly, embedded devices
Released	2010
Interpeted/Compiled	Compiled
Level	High-level

Questions?

JONATHAN VIJAYAKUMAR

Scala in a gist {}

> Functional & OO
> Strong Statically Typed
> Criticism-Fixed Java (to say..)

Scala in JVM

> Scala runs on JVM
> Inter-operability

Show Me Code First >

Hello World!

```
object syntax extends App {
    println("Hello, World!")
}
```

OO

```
class Stud(_id: Int, _stud_name: String) {
    var id: Int = _id;
    var stud_name: String = _stud_name;
}
object MainObj {
    def main(args: Array[String]) {
        var s = new Stud(10, "Jonathan")
        println(s.id + " " + s.stud_name);
    }
}
```

Class Hierarchy

> Scala.Any -> Parent of all

> Scala.AnyVal/Scala.AnyRef

> Scala.Null -> null (cannot assign

Operations

> Everything is a function, even a +

```
def + (x: Nat): Nat =
    if (x.isZero) this else succ + x.pred
```

Variables and Properties

> Var (Imm.) / Val (Mut.)

```
def var_name: String
def var_name_ = (newval: String): unit
```

```
var_name = "some-string"
```

Methods and Operations

```
def fName ([list of params]) : [return type] = {
    function body
    return [expr]
}

def arbFunction[T](arrayArb: Array[T]) =
{
    for (e <- arrayArb) println(e)
}
```

Nested Functions

```
def forall[T](xs: Array[T], p: T => boolean) = {
    def not_p(x: T) = !p(x)
    !exists(xs, not_p)
}
```

Comprehensions

> Short-cut to power

```
val genYield =
   for (std <- studList if std.age >=18)
     yield std.id
```

Abstraction (Parameterization)

```
class ArbClass[T](stud: T) {
    var student: T = stud
}
def swap[T](stud1: ArbClass[T], stud2: ArbClass[T]): unit = {
    val t = stud1.stud; stud1 = (stud2.stud); stud2 = t

    val stud1: ArbClass[int] = new ArbClass[int]("jonathan")
    val stud2: ArbClass[int] = new ArbClass[int]("david")
```

Abstraction (Abstract Members)

Abstract Class / Virtual Methods

```
abstract class AbsClass {
  type T
  val name: T
  def print_name(): unit = { println("Hello " + name) }
}

val absImpl = AbsClass { type T = String; val name = "Jonathan"}
absImpl.print_name()
```

Compositions

Functionality chaining

```
val add = (num: Int) => {num + 1}
val sub = (num: Int) => {num - 1}

println((add compose sub)(5))
```

```scala
trait Animal {
    def region: String
}

class Bear(val region: String) extends Animal

val anml = new Bear("Canada")

trait Animate extends Bear {
    def animateAnimal(word: String) = {
        println("Bear says: " + word)
    }
}

val anmlMixin = new Bear("Canada") with Animate
anmlMixin.animateAnimal("Hello")
```

OO -> Classes to Models

```
abstract class Person {
  val name: String
}

abstract class Employee extends Person {
  val id_number: Int
}

abstract class CEO extends Employee {
  val cabin_number: String
}
```

What is Scala for?

General Purpose

Who is Scala for?

Everybody + Java/C# Devs

Scala Philosophy?

To boost component-based SW

Is Scala hard?

Well, Yes and No

scala-lang.org

Introduction to Scheme

Enea Risto

Agenda

- History
- Characteristics
- Variables & Expressions
- Motivation
- Conclusion

LISP

History

1975	1995	2005	2010
Scheme	R5RS	R6RS	R7RS small

Characteristics

functional language

parenthesized prefix notation

Characteristics

(operator operand_A operand_B)

Hello World

```
;Hello World Program.

(begin
  (display "Hello, World.")
  (newline))
```

Simple Data Types

booleans

numbers

characters

symbols

Compound Data Types

strings

vectors

dotted pairs

Variables

```
(define x 5)
(define add7 (lambda (x) (+ x 7)))
x          => 5

(add7 3) => 10
```

Expressions

literal expressions

"xyz" => "xyz"

variable references

(define x 18)

Expressions

lambdas

```
(lambda (x) (+ x x)) => a program
```

conditionals

```
(if (> 1 4) 'yes 'no) => no
```

Motivation

the Actor model

Motivation

the academic usefulness

Conclusion

Python

Concepts of Programming Languages

Hochschule für Technik Stuttgart
University of Applied Sciences

Sonali Hanamantache

Agenda

- What is Python?
- History
- Evolution
- Structure of Python Program
- Basic Syntax
- Features
- Advantages and Disadvantages

What is Python?

- Python is a general purpose programming language.
- It is high level, object-oriented programming languages which runs equally on different platforms such as:
    - Windows, Mac OS, Linux etc.
- Python has a set of standard libraries therefore it also described as batteries included language.
- Dynamically Typed.

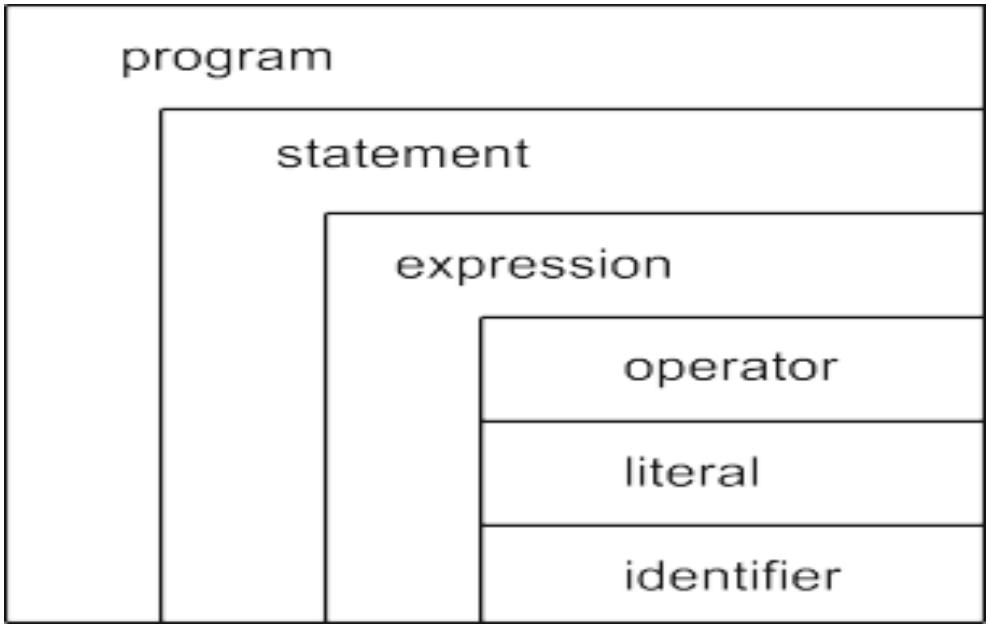
History

- Python was conceptualized by Guido Van Rossum in the late 1980's.
- Python implementation was started in 1989 and named after famous TV show Monty Python's Flying Circus in Netherland.
- Python was an alternative to ABC programming language, which is a general purpose programming language that has been developed at the CWI.
- It was first released in 1991 as Python 0.9.0.

Evolution

- Python 2 was released in 2000 with new features like
  - List comprehension
  - Garbage collection using reference counting
  - Support for Unicode
- Python 3 was released in 2008 and it was unique release as it was not backward compatible.
- Python 2's *print* statement has been replaced by the *print()* function in Python 3.

Structure of Python Program



By Y. Zhang (2015). An Introduction to Python and Computer Programming.

Basic Syntax

- **Indentation** used in Python to delim
- The header line for compound state
  like *if, while, def* should be termina
  colon (**:**)
- Printing to the screen
  - **print("Hi")**
- Reading keyboard input
  - **name=(input("Enter name"))**
- Comments

  For single line - **#Single line comment**

  For multiple lines - **'''**

  Multiple line comment

  Multiple line comment

  '''

```
x =int(input("Please enter an integer: "))
if x < 0:
  x = 0
  print("Negative changed to zero")
elif x == 0:
  print("Zero")
elif x == 1:
  print("Single")
else:
  print("More")
```

Features

- Simple and Easy

```
print('Hello, world!')
```

- High-Level Language
- Open Source
- Portable
- Object Oriented
- Interpreted
- Standard Libraries
- Functional Programming

Advantages and Disadvantages

- Advantages
  - Availability of third party modules
  - Open source and Community development
  - Portable

- Disadvantages
  - Python code errors only visible at runtime
  - Execution is slower than C , C++
  - Less suitable for mobile development

Thank You!!

JAVASCRIPT

CONCEPT OF PROGRAMMING LANGUAGES

FURQAN IQBAL

AGENDA

- Introduction
- usage
- Client- side
- Event Handling
- Variable
- Function
- DOM

WHAT IS JAVASCRIPT ?

- Client side interpreted lightweight programming language used to make web pages interactive.
- insert dynamic text into HTML (ex: user name)
- react to events (ex: page load user click)
- used to enhance websites.
- Can manipulate HTML

TO USE OR NOT TO USE

Helpful for:

- Dynamic content
- Adding logic to HTML
- Make changes without refreshing the page
- Form validation
- Ease processing on server

Not helpful for:

- Accessing resources
- Do anything that requires privacy
- JavaScript is shown publicly

CLIENT-SIDE SCRIPTING (JAVASCRIPT) BENEFITS:

usability:

- can modify a page without having to post back to the server (faster UI)

efficiency:

- can make small, quick changes to page without waiting for server

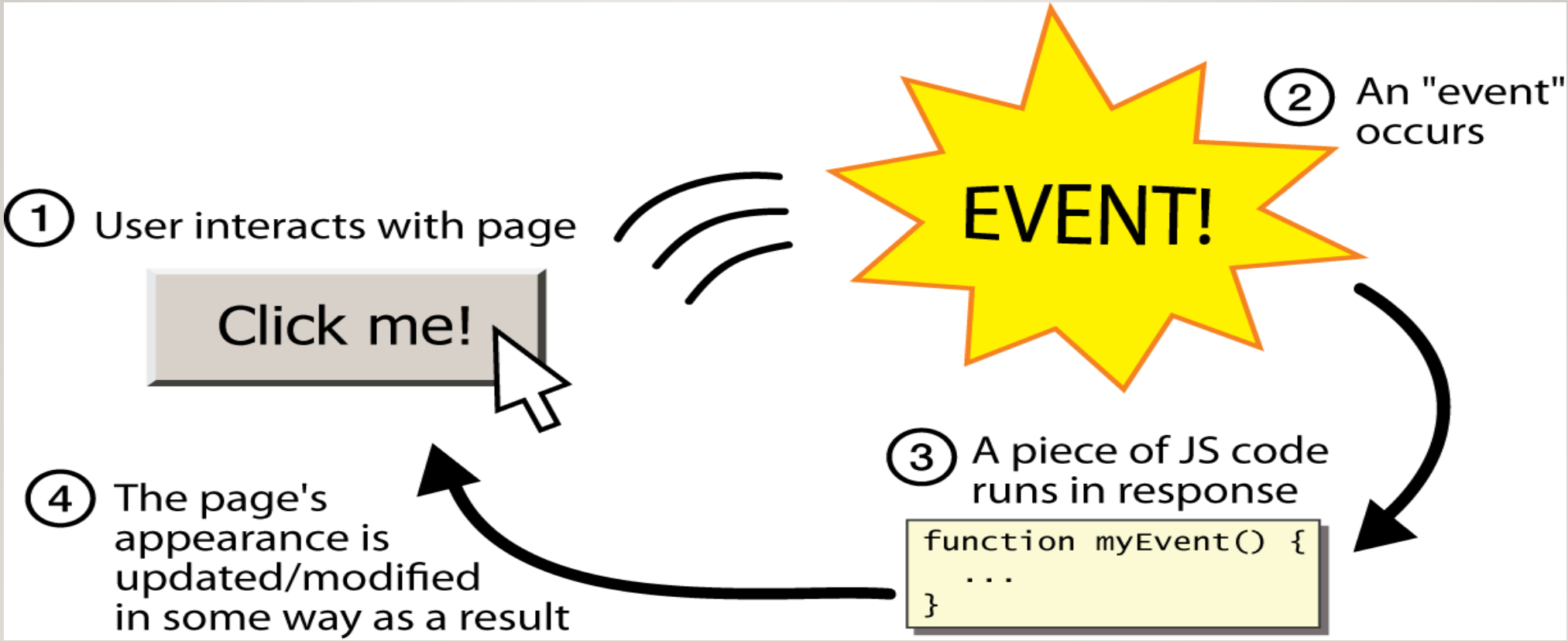
event-driven:

- can respond to user actions like clicks and key presses

EVENT HANDLERS

- JavaScript functions can be set as event handlers
- when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- but popping up an alert window is disruptive and annoying
- A better user experience would be to have the message appear on the page...

EVENT-DRIVEN PROGRAMMING



VARIABLES

- Variables are untyped
- No need to declare type (int, bool, etc)
- Doesn't care about type until called

Example:

- var name = "Andrew";
- var num = 24;
- var name = num;
- name will now return 24

FUNCTIONS

- Similar to any other languages' functions and methods
- Can have nested functions
  - Variables have scope similar to nested loops
- Used by events
  - Runs only when called
- Example:
  - function <name> (<parameters>) { \\code }

ADDING JAVASCRIPT TO HTML

- Inline:
- <script language="javascript"> \\code </script>
- External:
- <script language="javascript" src="code.js" />
- Typically this is placed in between the <head> tags
- Case sensitive (unlike HTML)

POPUPS

- Used for displaying information to the user outside of the page.
  - Typically as a warning or when an error has occured
  - Ask user for additional information
  - Confirmation
- alert(<text>); - Exits via an okay button
- confirm(<text>); - Returns true or false (ok or cancel)
- prompt(<text>, <default>); - Returns user's input

DOCUMENT OBJECT MODEL (DOM)

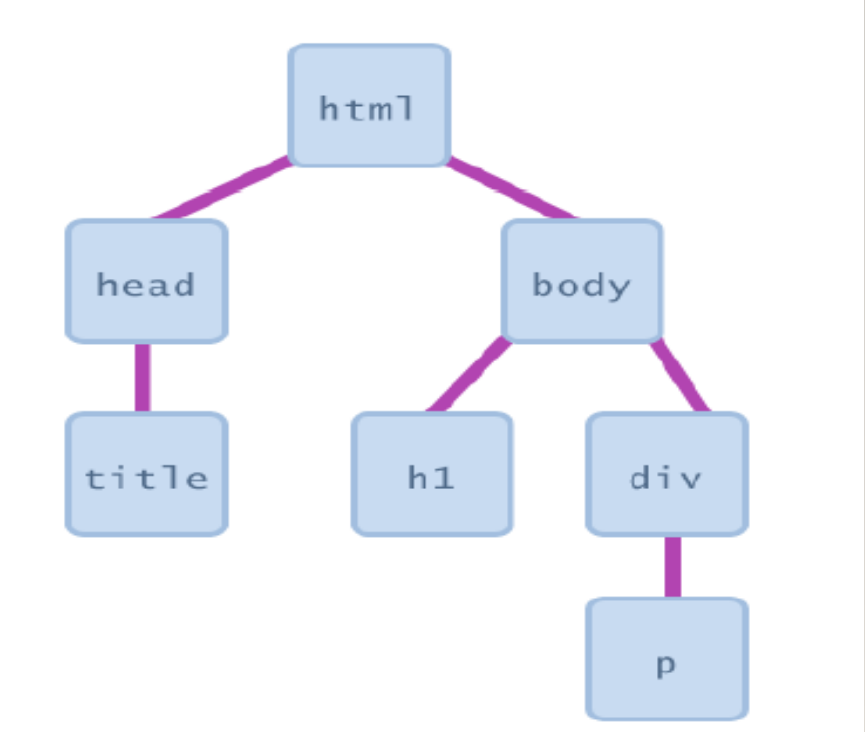
- most JS code manipulates elements on an HTML page
- The HTML DOM is an API (Programming Interface)

for JavaScript

- When a web page is loaded, the browser

creates a Document Object Model of the page

- we can change state
- e.g. insert some new text into a div
- we can change styles
- e.g. make a paragraph red



DOM ELEMENT OBJECT

HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

Thank you for attention

m4 Macro language

CPL SUMMER 2021

RUTUJA ANANT PAWAR

Agenda

- Introduction
- History
- Programing Structure
- Features
- Application
- Conclusion

Introduction

- **m4** is a general-purpose macro processor included in most Unix-like operating systems.
- m4 language transform one source file into other, hence its an Interpreted Language
- Implementation of m4 is mostly done in Operating Systems.
- m4 supports the following distinguishing features:
  - **a free - form syntax**
  - **high degree of macro expansion**

History

- Macro processors became popular when programmers commonly used assembly language.
- It is an extension of an earlier macro processor **m3.**
- Mr. Kernighan and Mr. Ritchie developed m4 in 1977
- GNU m4 was taken over by Mr. Francois Pinard in 1992, and he maintained it until 1994, when he released GNU m4 1.4, which was the stable version for ten years

1. Defining Macros

- A macro description is made up of a symbolic name (referred to as a token) and the character string that will be used to replace it

- The **define** built-in is used to define macros.

- Consider the following scenario:

  define (R,100)

  define (M,R)

2. Variables

- Macros is treated as (text) variables.
- macro names in m4 are global.
- Consider the following example:

  define(‘Count’,‘incr(Count)’)

1. Higher order macros

- Higher-order macros are macros that take macros as parameters.
- On the call, the macro must be quoted as a parameter.
- Apply can be used for unary and binary operators.
- Consider the following example:

  define(Apply,$1($2,$3))

  This allows for operations with one or two parameters, such as:

  Apply('Average',23,9) Average(23,9)

2. Construct macros with macros

- macros can create other macros
- Consider the following example

  defineblind (name, [value])
- If these characters “$#” are adjacent in the specification of define blind, the number of arguments to define blind will be extended instead of the number of arguments to call.

Features of m4

Some other featured offered by m4:

- file inclusion
- string manipulation
- conditional evaluation
- arithmetic expressions
- system interface

Application

- M4 is used as a preprocessor by some device programmes, such as sendmail, and application programmes, such as FrameMakerTM.
- m4 can be used to build a more friendly LaTeX-compatible picture-drawing syntax defines a macro package that allows you to draw different shapes.
- Its main purpose has been to serve as a front end for Ratfor.

Conclusion

- m4 can be used to provide syntactic sugar for a language and to facilitate conversion between languages.
- The m4 has been used in the development of OSI specifications, the production of telecommunications services, and the design of digital logic circuits.
- m4 currently do not support dynamic module loading, debugging is difficult.
- m4 emphasizes on readability, cost, portability, standardization, correctness.

Summary

Designed by	Mr. Kernighan and Mr. Ritchie
Paradigm	Macro
Released	1977
Compiled/Interpreted	Interpreted
Features	• a free - form syntax • high degree of macro expansion
Implementation	GNU m4, FreeBSD, OpenBSD, NetBSD, Inferno OS

References

- 1] GNU m4 manual, online and for download in HTML, PDF, and other forms. ”GNU M4 — GNU macro processor”, accessed January 25, 2020.
- [2] The M4 Macro Processor, Brian W. Kernighan, Dennis M. Ritchie, Bell Laboratories, Murray Hill, New Jersey

THANK YOU

Concepts of Programming Language

Summer 2021

MATLAB

Priya Yuvaraj

M.Sc Software Technology
Hochschule für Technik,Stuttgart
11yupr1mst@hft-stuttgart.de

Agenda

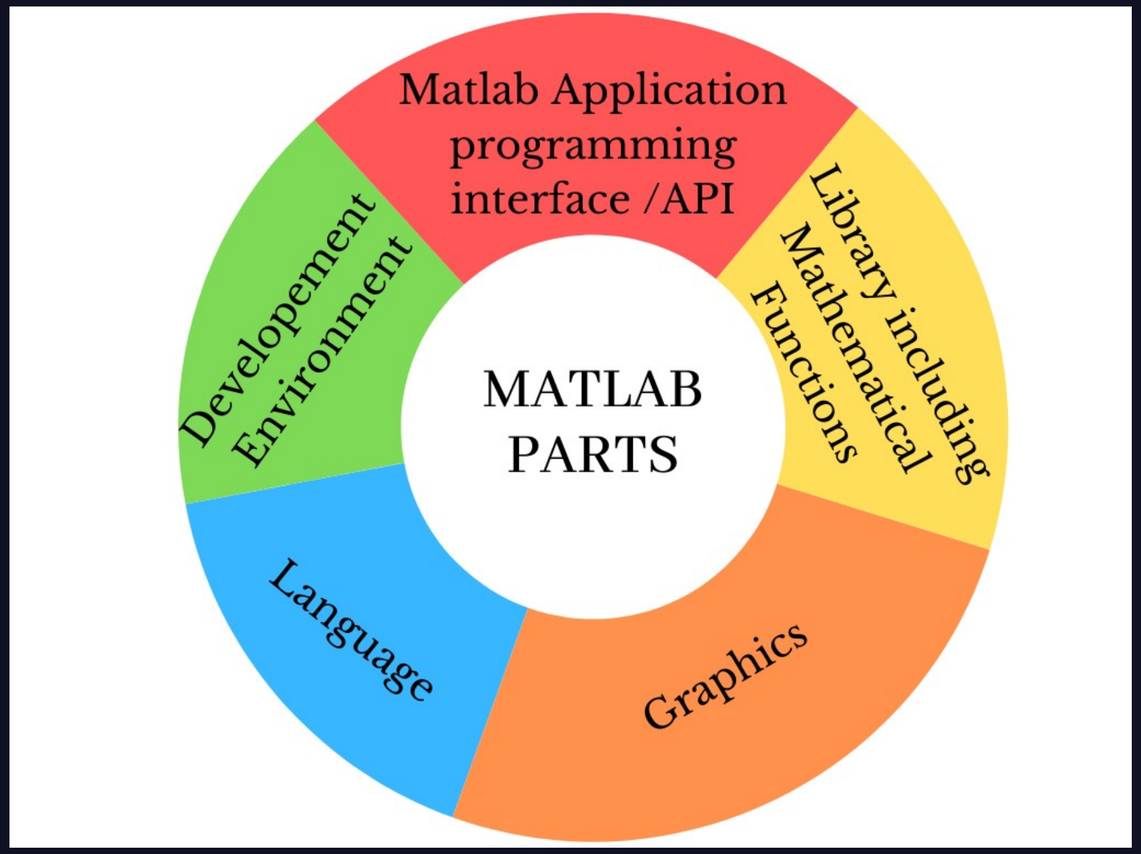
Points for discussion

- What is MATLAB?
- Why was MATLAB Developed?
- Why do we need MATLAB?
- Where MATLAB is used?
- Impressive Programming Aspects
- Simulink
- Differences and Drawbacks
- Guidelines and Impact

Priya Yuvaraj
Hochschule für Technik,Stuttgart

What is MATLAB?

- MATLAB is a term for Matrix Laboratory.
- High level programming language and provides its own IDE and library package.
- Vector - oriented.
- Interfacing with C, C++, Java, and FORTRAN are all possible with Matlab.



Priya Yuvaraj
Hochschule für Technik,Stuttgart

Why was MATLAB Developed?

Prof. Dr. Cleve Moler, the Computer Science Department Chairman at the University of New Mexico, was initially designed MATLAB in late 1970.

To accomplish linear algebra and numerical computations without utilizing Fortran.

MathWorks was headquartered in 1984 in California by Cleve Molar, Steve Bangart, and Jack Little.

Priya Yuvaraj

Hochschule für Technik,Stuttgart

Why do we need MATLAB?

Logical systems can rapidly and efficiently have implemented using the MATLAB software framework.

MATLAB primarily solves numerical problems, and it entails engineering, scientific measurements, and simulations.

Blocks for modeling equations of motion, navigation, gain scheduling, visualization, unit conversion.

With just a few lines of code, users can program or simulate.

MATLAB helps to implement and formulate various algorithms.

Priya Yuvaraj
Hochschule für Technik,Stuttgart

Where MATLAB is used?

Central analytical platform in the science and engineering field.

Promising potential in the automotive, medical, and aerospace domains.

Collaborative system for sequential problem solving, architectural design, and innovation.

Priya Yuvaraj

Hochschule für Technik,Stuttgart

Impressive Programming Aspects

Double and char are two common types of MATLAB variables.

The colon(:) is amongst the most beneficial operators. It helps to generate vectors, subset arrays and validate over each.

Set of built-in functions to make the programmer's work easier.

There are several fundamental data types in MATLAB. Especially cell array, character array, structure, map containers, and function handles.

Priya Yuvaraj

Hochschule für Technik,Stuttgart

Impressive Programming Aspects

High-level commands for 2D and 3D visualization are obtainable in Graphics.

MATLAB's interactive spectrum extended in a variety of ways. Namely, Financial Toolbox, Image Processing Toolbox, Neural Network Toolbox, Optimization Toolbox, Partial Differential Equation Toolbox, Signal Processing Toolbox.etc.

Simulink

Simulink is a visually impressive package for modeling and analyzing dynamic systems.

Great support in satellite, aircraft, spacecraft, and image processing, as well as automotive systems.

In data analysis, Simulink promises the GUI for building models as diagrams.

Difference

For statistical computation, MATLAB outperforms Excel.

Highly suitable for analyzing numbers, arithmetic, calculus, and several other obstacles.

Perform symbolic computations like simplifying equations, which Fortran cannot.

Drawbacks

Low speed: interpreted language, it is slower than a compiled one.

Cost: economical for industry.

Priya Yuvaraj
Hochschule für Technik,Stuttgart

Impact

In 2012, IEEE recognized the implications of Dr. Moler's contribution by honoring him with the Computer Pioneer Award.

Today in 2021, it has over 1 million users worldwide and 100 products based on MATLAB. Almost taught in more than 5,000 universities.

Guidelines

Professional customizing documentation offers by technologists.

MATLAB has its own Help Center on the Mathworks website.

Priya Yuvaraj
Hochschule für Technik,Stuttgart

Conclusion

MATLAB is an interpreted and multi paradigm language.

Dynamic - Weakly typed language, handle type conversion.

Parallel Computing, diverse selection of maths application areas, and it correlates to the dynamic system simulation package.

These kinds of qualities have made Matlab a tremendously valuable platform.

Priya Yuvaraj
Hochschule für Technik,Stuttgart

Thank you
for your time
and
attention!

Priya Yuvaraj
Hochschule für Technik,Stuttgart

Pascal Programming Language

Concepts of Programming Languages
Student Presentation

Harini Srikanth – (1001715)

Innovative Hochschule
Eine gemeinsame Initiative von Bund und Ländern

Agenda

- Introduction
- History
- Wirthian Perspective
- Pascal Standards
- Pascal usage and purpose
- Pascal Building blocks
- Pascal Language elements
- Criticizations on Pascal
- Compilers of Pascal

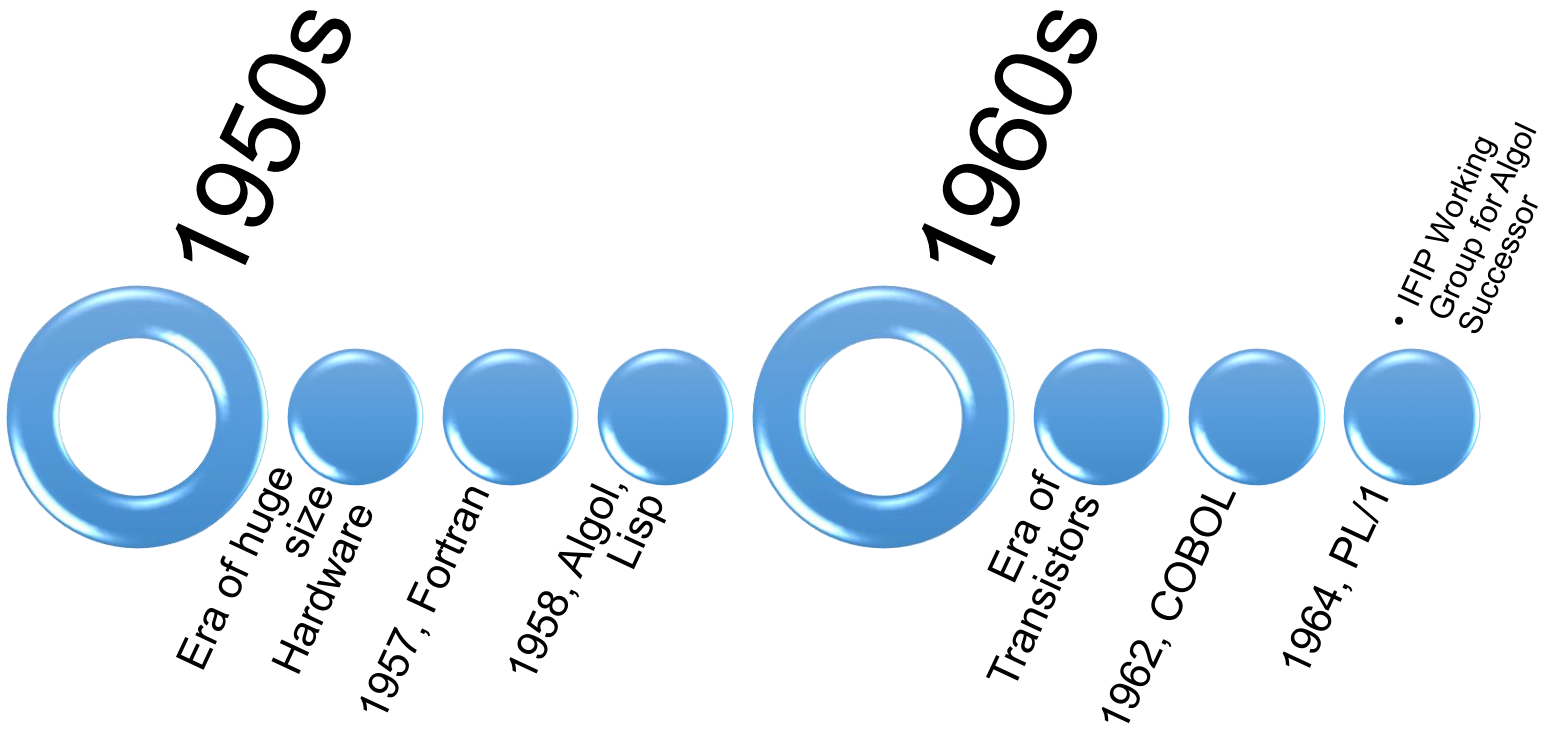
Agenda

- Pascal Today – FreePascal, Delphi
- Pascal Today – PLC Structured Text
- Key Takeaways
- References
- Questions

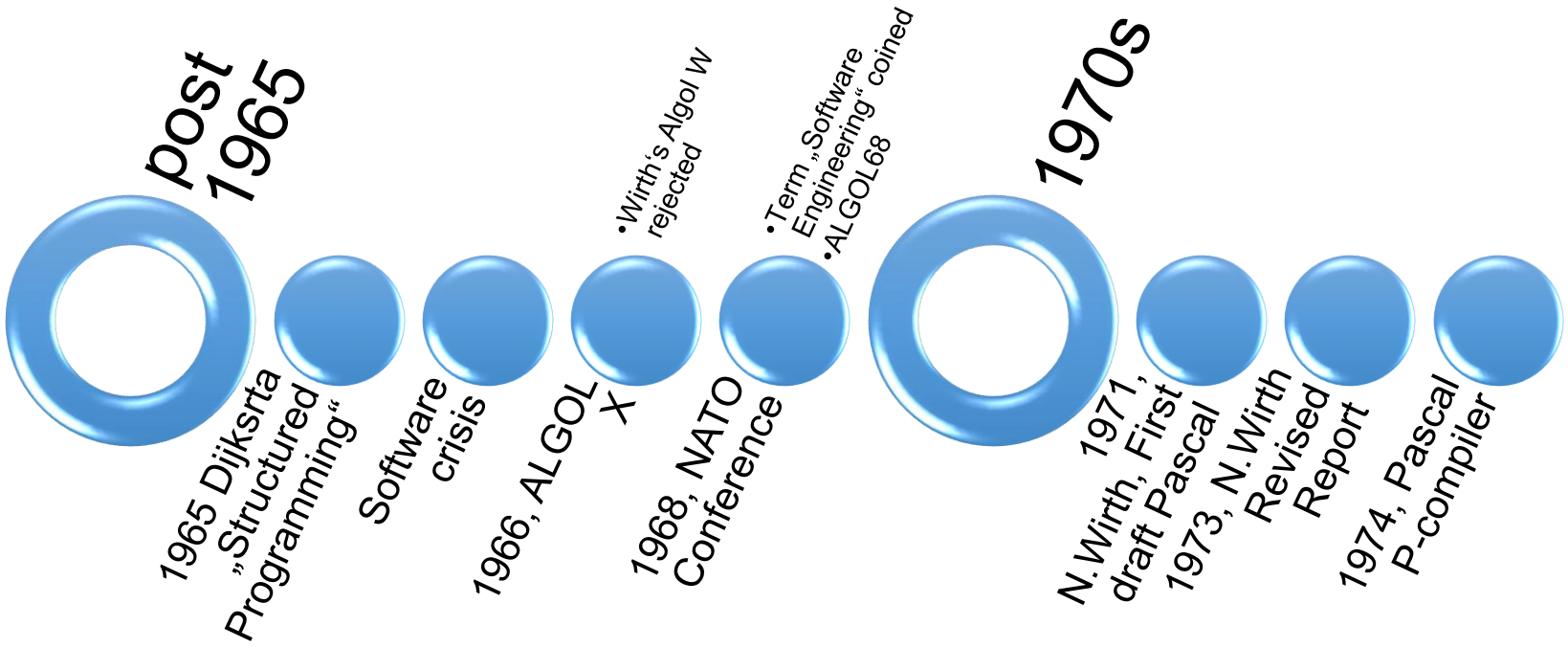
Introduction

- Introduction
  - Founder – Prof.Dr.Niklaus Wirth, ETH Zurich,published in the year 1971
  - Named after French Mathematician Blaise Pascal
  - Imperative, Procedural, strong-typed, structured programming
  - Pascal – classical programming language for scientific education
  - Derived from (FORTRAN), ALGOL and COBOL
  - Features ahead of its age with records, sets, files, arrays
  - Insights from its designer and uniquesness of the compiler
- Influenced ADA, Modula and Oberon

HISTORY AND TIMELINE



HISTORY AND TIMELINE



Pascal Standards

- 1974, Prof.Dr. N. Wirth, Revised Report
- Attempts on Standards for Pascal
  - 1977-78, BSI from UK poposal for Pascal International Standards
  - 1978-79, USA IEEE Project 770 for Pascal
  - 1979, ISO Working Group for Pascal formed
  - 1982, BS6192 from BSI
  - 1983, ANSI → ANSI/IEEE770X3.97-1983
- 1985, ISO 7185, Unextended Pascal
- 1990, ISO 10206 → Extended Pascal additions
  - **,><,
  - And_Then, Bindable, Export, Qualiied, Restricted, Protected, Only, Or_else, otherwise, value, Module, POW etc
- Several other implementations are famous

Wirthian Perspective

- Importance of systematic and structured programming right at inception during studies academics
  - *“A good designer must rely on experience, on precise, logic thinking; and on pedantic exactness. No magic will do.”*
- Emphasizes simplicity
  - *Reliable and transparent programs are usually not in the interest of the designer. The wishes of the users count more than their needs*
- Is Programming easy ?
  - *“It is not programming in the sense of coding that is the problem, but design. Not every programmer is a born designer.”*
  - *“Many people tend to look at programming styles and languages like religions: if you belong to one, you cannot belong to others. But this analogy is another fallacy”.*
- Turing Prize – 1984
- Wirth‘s Law – „Software gets slower more rapidly than Hardware becoming faster“
- Extended the learnings from Pascal and its shortcomings in Oberon and Modula-2

Purpose and Usage

- Pascal primary aims
  - Academically correct language for teaching programming fundamentals
  - state-of the art efficient and reliable programming language then
- Pascal usage Examples
  - Apple Lisa
  - Mac OS
  - TeX
  - Adobe Photoshop 1.0
- Pascal successors Delphi, Object Pascal, FreePascal
  - Skype UI
  - Beyond Compare
  - Total Commander

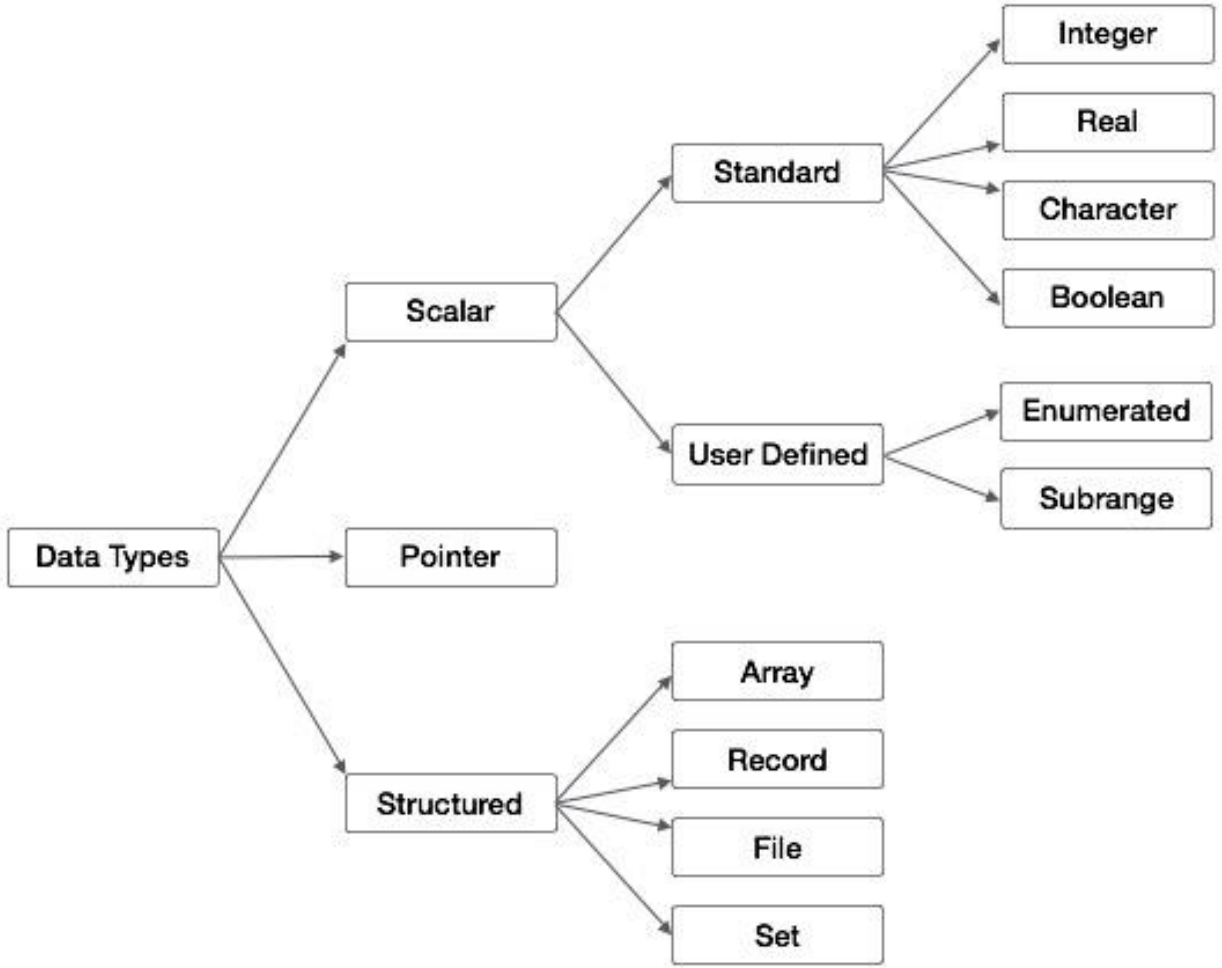
Building Blocks of Pascal

- Basic Structure
  - Strong typing, case-insensitive, verbose
  - forward declaration
- Unxextended Pascal keywords
- Semicolons are statement seperators
- Comments {...} or (*...*)
- Constants –
  - **const** keyword
  - **eol** is standard constant identifier
- Assignment „:=„ and not „=„
  - Like in C, i=j=0; not possible

```
letters  = 'a..z'and 'A..Z'
digits = '0..9'
special-symbol =
'+' |'-'|'*'|'/'|'='|'<'|'>'|'['|']'|
'.'|','|':'|';'|'"'|'^'|'('|')'
'<>'|'<='|'>='|':='|'..'|word-symbol
word-symbol=
'and'| 'array'|'begin'|'case'|'const'|'div'
|'do'|'downto' |'else'|'end'|'file'|'for'
|'function'|'goto'|'if'|'in'|'label'|'mod'
|'nil'|'not'|'of'|'or'|'packed'|'procedure'
|'program'|'record'|'repeat' |'set' |'then'
|'to'|'type'|'until'|'var'|'while'|'with'
```

Pascal Data types

- Basic data types → Scalars
  - Integer, Real, Char, Boolean
- User defined datatypes
  - Subrange, Enumeration
- Collections
  - Array, Record, File, Set
  - Sets support Union, Intersection, Difference and membership operations



Basic Examples

Enumerations and Subrange

```
Type
  Number = 1..100;
  Value = P..Q;{subrange}
  Fruit = (apple, banana, citrus);{enum}
```

Hello World

```
program HelloWorld(output);
 begin
 WriteLn('Hello world!'){if semicolon
    here --> null statement}
 end.{don't forget period symbol at end}
```

Sets

```
var
Set1 : set of 1..10;
Set2 : set of 'a'..'z';
{Advantage of sets: Use of constructs
   like}
if i in set1 then ...
{than typing ...}
if (i > 4) and (i < 11) then ...
```

Arrays

- Arrays are collections of same datatype elements
- Indexed. Assignment through loop.
- Speciality:
  - Index types can be defined
  - Start value need not always be 1 or 0 or integer as in other languages

Programming language	Default base index	Specifiable index type[13]	Specifiable base index	Bound check	Multidimensional	Dynamically-sized	Vectorized operations
Objective-C[16]	0	no	no	checked	array of array[22]	yes	no
OCaml	0	no	no	checked by default	array of array[22]	init[15]	?
Pascal, Object Pascal	index type[14]	yes	yes	varies[41]	yes	varies[42]	some
Perl	0	no	yes ($[ )	checked[35]	array of array[22]	yes	no[43]
Raku	0	no	no	checked[35]	yes	yes	yes
PHP	0	yes[44]	yes[44]	checked[44]	yes	yes	yes
PL/I	1	?	yes	checked	yes	no	?
Python	0	no	no	checked	array of array[22]	yes	no[45]
RPG	1	no	no	?	no	no	?
R	1	?	?	?	?	?	?
Ring	1	?	partial[36]	checked	array of array[22]	yes	?
Ruby	0	no	no	checked[35]	array of array[22]	yes	?
Rust	0	no	no	checked	yes	?	?
Sass	1	no	no	checked	array of array[22]	init[30]	?
S-Lang	0	?	no	checked	yes	yes	yes
Scala	0	no	no	checked	array of array[22]	init[15]	yes (map)
Scheme	0	?	no	checked	array of array[22]	init[15]	yes (map)
Smalltalk[16]	1	?	no	checked	array of array[22]	yes[46]	?
Swift	0	no	no	checked	array of array[22]	yes	?
Visual Basic	0	no	yes	checked	yes	yes	?
Visual Basic .NET	0	no	partial[23]	checked	yes	yes	yes (LINQ select)
Wolfram Language	1	no	no	checked	yes	yes	yes
Windows PowerShell	0	no	no	checked	yes	heap	?
Xojo	0	no	no	checked	yes	yes	no
XPath/XQuery	1	no	no	checked	array of array[4][22]	yes	yes

Language Elements: memory referencing

- Pointers
  - refers to the address of variable denoted by ^ or @
  - powerful combination with records → linked lists, stacks, queues
  - type-safe
  - new()...dispose()
  - nil value is same as null
- Files are sequence types with components primitive or predefined type
  - Buffers – file pointers to access components
  - Lookahead function
  - Procedures- **reset, rewrite, put, get, close**
  - **eof, eoln**
- Packed
  - provision for packing the collections with advantage of storage compromising access time
- Variants
  - provision for Unions in Pascal

Language elements: Control Structures

- Conditional Statements
  - If..Else - used **begin..end** for blocks inside
  - Switch case -**no default** constructs
  - Goto with declared labels (0..9999)
- Loops
  - Repeat..Until – similar to do..while in C
  - While loop
  - for – provisions for upcount with **TO** and downcount with **DOWNTO**
  - **No continue, break** statement → **Goto** or exit only after loop completion

```
case Nr of
0 : Write('the nr is one');
1 : Write('the nr is two');
2 : Write('the nr is three');
3..10: Write('I dont know') {no default
   switch is available !}
end;
```

```
{example of nested-if}
if b > 0 then
 begin
  if c = 3 then m := TRUE
  else
   begin
    c := 3;
    m := FALSE;
   end
 end
else if B < 0 then m := FALSE
else
 begin
  c := 1;
  m := FALSE;
```

```
{while example}
while a AND (count <= max) DO
begin
 count := count + 1;
 val := 2 * val;
 a := val < 100;
END;
{increment for}
for i := 1 to 10 do  {no semicolon for
   single statements allowed!}
WriteLn('Counting: ', i);
{decrement for}
for B := 50 downto 0 do
d := d + 2 * b;
{example of repeat..until}
repeat
 myval := myval + 1
until myval = 10;
```

Language Elements: Subroutines, modules

- Program is the outermost block – comparable to main() in C
  - Variables here are of **global** scope
- Procedures
  - similar to scripts or set of statements. **No return value !**
  - read(..) and write(..) are standard procedures
  - has an **identifier** and optionally **parameters** and **local** variable declaration possible
  - Nesting procedures possible
- Functions
  - **return** a single scalar **value,** defined in declaration
  - **No return** statement
  - **local** temporary variable declarations
- Standard Functions,Procedures

Pascal Compilers

- Pascal P4/P5
  - first set of Portable Pascal implementations
- Turbo Pascal
  - popular and efficient Pascal compiler from Borland
- GNU Pascal (GCC)
  - written in C
- Dr. Pascal
  - interpreter with visible runtime error checking for Windows DOS, old Mac OS etc

Pascal Today

- Free Pascal
  - Multiplatform compiler, self-hosting
  - strong community, blogs and documentation
  - Lazarus IDE
- Delphi
- Critics
  - Brian Kernighan „Why Pascal is not my favorite Programming language“

Pascal and PLC Structured Text (ST)

- IEC-61131-3 ST
- Similarities
  - Assignment :=
  - control structures and forward declaration
  - Program, Functions
- Differences
  - No Recursion in ST
  - Function Blocks in ST
  - default initial values

Structured Text	Pascal

`FOR VarB := 1 TO VarC DO`
`  VarD := VarD + 1;`
`  VarE := 2 * VarE;`
`END_FOR;`
	`FOR VarB := 1 TO VarC DO`
	`BEGIN`
	`  VarD := VarD + 1;`
	`  VarE := 2 * VarE;`
	`END;`

`FOR VarB := 100 TO 0 BY -2 DO`
`  VarD := VarD + VarB;`
`END_FOR;`
	`FOR VarB := 50 DOWNTO 0 DO`
	`  VarD := VarD + 2 * VarB;`
	`END_FOR;`

`WHILE VarA AND (VarD <= VarC) DO`
`  VarD := VarD + 1;`
`  VarE := 2 * VarE;`
`  VarA := VarE < 100;`
`END_WHILE;`
	`WHILE VarA AND (VarD <= VarC) DO`
	`BEGIN`
	`  VarD := VarD + 1;`
	`  VarE := 2 * VarE;`
	`  VarA := VarE < 100;`
	`END;`

`REPEAT`
`  VarD := VarD + 1;`
`  VarE := 2 * VarE;`
`  VarA := VarE < 100;`
`UNTIL NOT(VarA AND (VarD <= VarC));`
	`REPEAT`
	`  VarD := VarD + 1;`
	`  VarE := 2 * VarE;`
	`  VarA := VarE < 100;`
	`UNTIL NOT(VarA AND (VarD <= VarC));`

Structured Text	Pascal

`FUNCTION Ave_REAL : REAL`
`  VAR_INPUT`
`  Input1, Input2 : REAL;`
`  END_VAR`
`  Ave_REAL := (Input1 + Input2) / 2;`
`END_FUNCTION`
	`FUNCTION Ave_REAL`
	`  (Input1, Input2 : Real): Real;`
	`BEGIN`
	`  Ave_REAL := (Input1 + Input2) / 2;`
	`END;`

Key Takeaways

- A good starting point to learn Programming other than C
- Advent of DIY kits and Self-taught systems like Robotic kits and microcontrollers – Arduino, Raspberry Pi
- A good look at Prof.Wirth‘s work and his excerpts show the immense work behind a Programming language evolution
- Pascal was ahead of its age with provisions for set, files, records. Its elements had been carried forward to other languages

References

1) N. Wirth, Recollections about the Development of Pascal. New York, NY, USA: Association for Computing Machinery, 1996, p. 97–120
2) N. Wirth, “The programming language pascal,” Acta informatica, vol. 1,no. 1, pp. 35–63, 1971.
3) N. Wirth, “A plea for lean software,” Computer, vol. 28, no. 02, pp. 64–68, feb 1995.
4) N. Wirth, The Programming Language Pascal. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 515–543.
5) N. Wirth, “The programming language pascal (revised report),” Berichte der Fachgruppe Computerwissenschaften, vol. 5, 1973.
6) P. Mateti, “Pascal versus c : A subjective comparison,” in Language, Design and Programming Methodology, J. M. Tobias, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 37–69.

References

7) L. Goosen, “A brief history of choosing first programming languages,” in IFIP International Conference on the History of Computing. Springer, 2008, pp. 167–170.

8) N. Roos, “Programming plcs using structured text.”

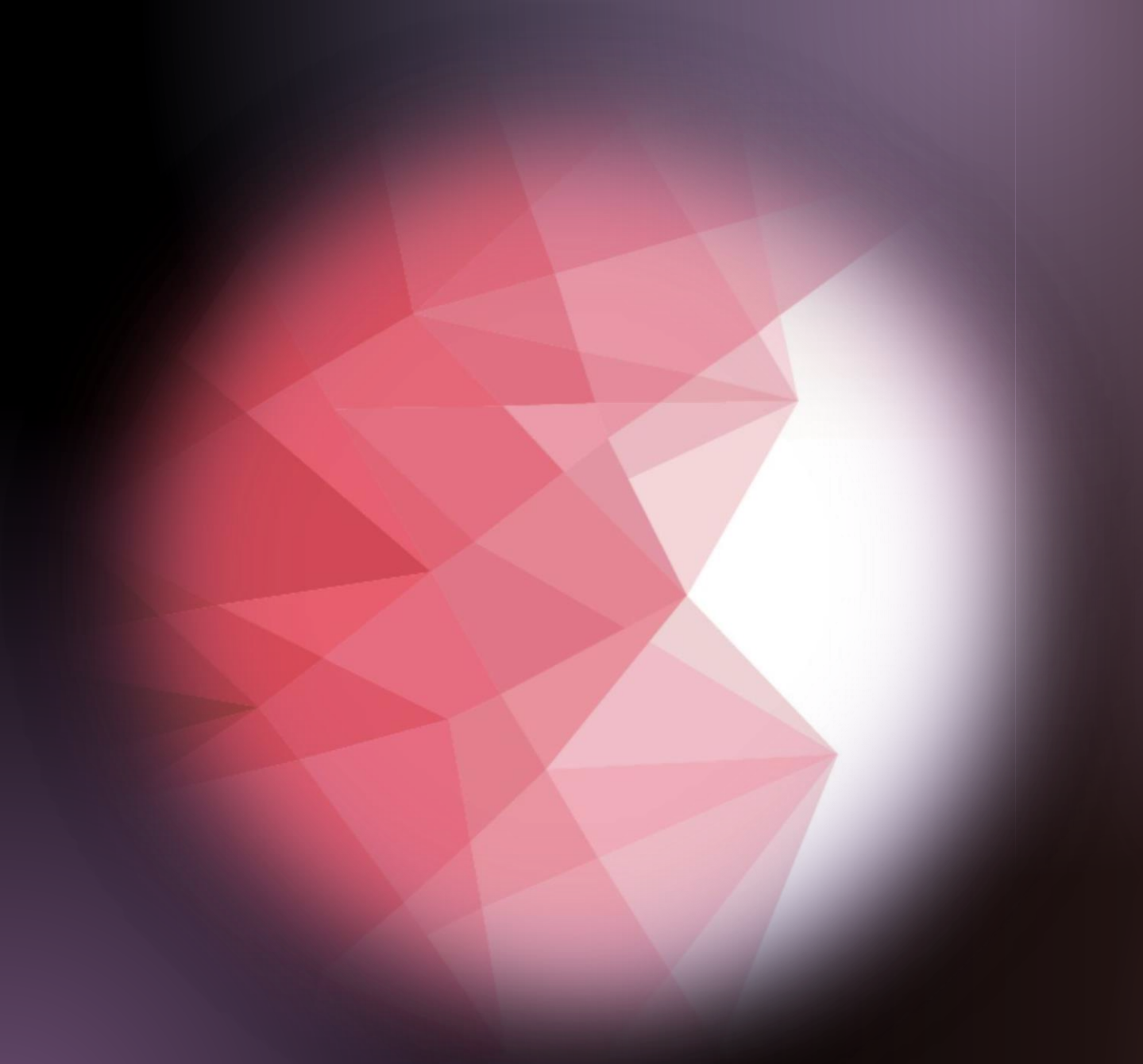
9) S. Sobral, “30 years of cs1: Programming languages evolution,” 11 2019.

10) K. Prof Dr.Blender, “Development of pascal, the first structured programming language,” https://www.eduard-rhein-stiftung.de/en/entwicklung-von-pascal-der-ersten-strukturierten-programmiersprache/, accessed: 2021-05-03.

11) Wirth, Niklaus. "A brief history of software engineering." IEEE Annals of the History of Computing 30.3 (2008): 32-39.

Questions

Thank you



Perl

THERE'S MORE THAN ONE WAY TO DO IT

Outline

- Introduction
- Hello World
- Variable Types
- Variable Interpolation
- Statements and Declarations
- Notable Differences
- Conclusion

Introduction

- Originally developed for text manipulation, but is now used for a wide range of tasks.
- Long since spread to most operating systems, despite being originally optimized for Unix.
- Particularly popular among web developers and system programmers but in fact it is much more widespread.
- Tisdall said: “If you can’t do bioinformatics, you can’t do biology, and Perl is the biologist’s favourite language for bioinformatics”.

Hello World

```
print "Hello World\n";
```

- The code above is the full code of the program, and if you feed it into the perl interpreter it will run properly.
- Code will exit normally after printing “Hello World” on the screen.
- Perl considers the end of a program a normal exit for the program.

Variable Types (Scalars)

```
$pi = 3.1415;
$pet = "Rabbit";
$result = 40;
```

- The $ sign identifies a scalar in Perl.
- A single variable is a scalar.
- Scalar variables can have any type of data assigned to them without having to explicitly specify the data type.

Variable Types (Arrays)

```
@array = ("element1", "element2", "element3");
```

- The @ sign identifies an array in Perl.
- A group of ordered scalars form an array.

Variable Types (Hashes)

```
%day = ("first", "Monday", "second", "Tuesday", "third", "Wednesday");
```

- The % sign identifies a hash in Perl.
- Constitute an unordered list of scalars with no beginning or end.
- Made out of a sequence of keys and values, one after the other.
- A value stored in a hash can be fetched by supplying its corresponding key.

Variable Interpolation

- Strings between double quotations are subject to variable interpolation, while those between single quotes are not.
- A common type of interpolation are backslashed character escapes (\n \r \t \b).
- Another type of interpolation is that of scalar variables and arrays.
- Scalars are inserted as they are:

```
$Price = '$200’;

print "A cellphone costs $Price.";

# output: A cellphone costs $200.
```

- Array are inserted with a space character separating each element:

```
@array = ("element1", "element2", "element3");
print "The elements are: @array.";
# output: The elements are: element1 element2 element3.
```

Statements and Declaration (1)

- Does not require variable declaration to be done before a variable is used.
- If and elsif statements:

```
if($a == $b){
        $value = 0xff0000;
}elsif($a == $c){
        $value = 0x00ff00;
}elsif($a == $d){
        $value = 0x0000ff;
}else{
        $value = 0x000000;
}
```

Statements and Declaration (2)

- The “while” loop serves to continuously execute a block of code as long as its condition resolves to true.
- The “until” loops in a similar fashion to a “while” loop but in the inverse sense.
- The “continue” block is executed in two cases, either by manually calling it from within the loop using the “next” operator, or by falling off the end of the loop.

```
while (my $line = <STDIN>) {
    $line = lc $line;
}
continue {
    print $line; # still visible
}
```

Statements and Declaration (3)

- In for loops, if you want to iterate through two variables simultaneously, just separate the parallel expressions with commas.

```
for ($i = 0, $bit = 0; $i < 32; $i++, $bit <<= 1) {
    print "Bit $i is set\n" if $mask & $bit;
}
```

- To change the scope of the created variables to only be accessible inside the “for” loop.

```
for (my ($i, $bit) = (0, 1); $i < 32; $i++, $bit <<= 1) {
    print "Bit $i is set\n" if $mask & $bit;
}
```

- “foreach” keyword can be replaced with “for” anytime without any impact on the execution.

Other Notable Differences

- Perl, unlike C++ and Java, doesn’t enforce privacy using keywords like private, protected, and public. If privacy is needed in Perl, lexical variables can be used.
- Perl does not require the constructor have the same name as the class. In Perl any method can play the role of a constructor. On the other hand, upon the destruction of a object, both Perl and C++ require well known destructor names.
- C++ and Java are statically typed languages, most of the type checking is performed during compile time, while Perl is a dynamically typed language, most of the type checking is performed during run time.

Conclusion

- We have looked at many of the main differences between Perl and object oriented languages like C++ and Java.
- Perl is a flexible and powerful programming language with many applications; it’s influence stretches to almost every industry and sector.
- Flexibility is Perl’s defining attribute.
- Perl is still relevant to this day after so many years since it was first released.

Thank You

Questions

References

- [1] M. J. Dominus, Higher-Order Perl: Transforming Programs with Programs. Elsevier, 2005.
- [2] L. Wall, T. Christiansen, and J. Orwant, Programming Perl, 3rd ed. 101 Morris Street, Sebastopol, CA 95472: O'Reilly Media, Inc, 2000.
- [3] J. Tisdall, Mastering Perl for Bioinformatics: Perl Programming for Bioinformatics. O'Reilly Media, Inc, 2003.
- [4] S. Srinivasan, Advanced Perl Programming. O'Reilly Media, Inc, 1997.
- [5] "Perl examples," https://www.asc.ohio-state.edu/lewis.239/Class/Perl/perl.html, accessed: 2021-04-25.

TeX

Concepts of Programming Languages, Summer 2021

Shamli Chellathurai Saroja, 11chsh1mst
Hochschule für Technik Stuttgart

AGENDA

- Introduction
- TeX Typesetting
- TeX Data Processing
- TeX Programming
- Advantages and Disadvantages
- Conclusion
- References

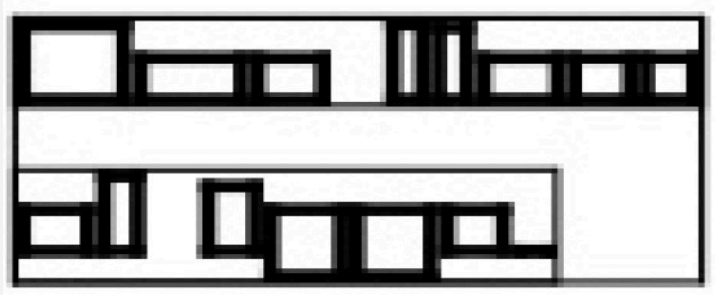
INTRODUCTION

- TeX, a typesetting system developed by Donald E Knuth, 1978
- Hot metal typesetting by a monotone machine used for Donald E Knuth's "The Art of Computer Programming"
- Monotone machines replaced by phototypesetting and deteriorated quality

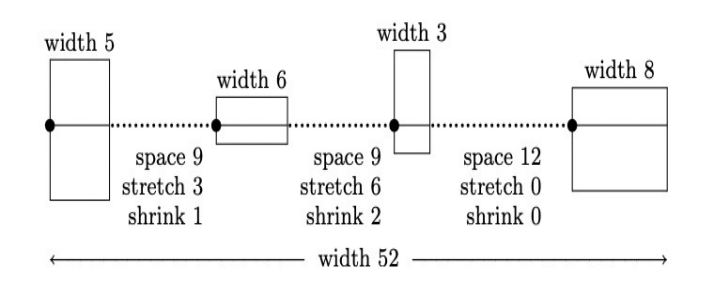
Two goals:

- Typesetting system consistent across all computers
- Create high quality books with minimal effort

TeX TYPESETTING



Boxes in TeX [1]



Glues in TeX [1]

- Every character is a **Box**; a word is a collection of boxes
- Words and lines are grouped together using **Glues**
- A line is a vbox; multiple vboxes inside a hbox forms a page
- **Ties, ~** are the secret to successful line breaking

Chapter~12, Theorem~1.2, Figure~3, Charles~III

DATA PROCESSING

- TeX coverts each input character to tokens of category codes

{\hskip 36 pt} : $\{_1 hskip3_{12}6_{12}10p_{11}t_{11}\}_2$

- Actual typesetting is performed on the tokens
- TeX generates device independent (**dvi**) output files containing final locations of all characters
- Control sequences (\hskip) are expanded to respective tokens

TeX PROGRAMMING: CONTROL SEQUENCES

- TeX uses control sequences for typesetting data
- Control sequences can be words or symbols

```
\bf, \it, \”, \$, \centerline, \backslash
```

```
\sqrt, \over, \choose
```

- TeX has control sequences that are specific to math formulas
- **Grouping, {}** determines how much of the text is to be influenced by certain control sequences

```
$$\sqrt{1+\sqrt{1+\sqrt{1+
\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$
```

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$$

TeX PROGRAMMING: MACROS

- Macros are user defined control sequences
- Avoids repetition of code, enhances reusability
- Macros can be defined with parameters and conditional statements

```
\def \row#1{(#1_1,\ldots, #1_n)}
\row{x}: (x_1,...,x_n)
\row{y}: (y_1,…,y_n)
\row{z}: (z_1,…,z_n)
```

```
\def \statement{\ifnum \balance=0 \fullypaid
\else \ifnum \balance>0 \overpaid
\else \underpaid
\fi
\fi}
```

TeX PROGRAMMING: LOOPS

- Macros can also be constructed to repeat actions in loop based on condition using **\loop** and **\repeat**
- \punishment macro with parameters to print ”I must stay focused” 100 times

```
\newcount \n
\def \punishment#1#2 {\n=0
\loop \ifnum \n<#2 \advance \n by 1
\item{\number\n.}#1\endgraf\repeat}
```

- Collection of macros form packages like LATeX, BIBTeX each targeting a specific document type

TeX ADVANTAGES & DISADVANTAGES

Advantages	Disadvantages
• Portability • Stability • Output formats • Content and styling separation • Templates • No voodoo fixes	• Constant learning

CONCLUSION

- Used for academic purposes, mathematic, scientific documentations, books, manuals
- Creates dynamic documents
- Makes documentation fun

REFERENCES

[1] Donald E. Knuth, ”The TEXbook”, Addison-Wesley PublishingCompany, 1992

[2] ”TeX. (2021)”. Wikipedia, the free encyclopedia, 04 May 2021,https://en.wikipedia.org/wiki/TeX

[3] E. Donald, Jill C. Knuth, ”TEX,” Encyclopedia of Computer Science,pp. 1756–1759, 2003

[4] ”LaTeX: when word fails you,” SIGUCCS ’05: Proceedings of the33rd annual ACM SIGUCCS conference on User services, pp.241–243, 2005

THANK YOU

Hochschule für Technik Stuttgart

Visual Basic

By

Edem Koffi Avode

Contents:

- Introduction
- Syntaxes
- OOP Features
- Uniqueness
- References

Introduction:

"Visual Basic is programming language developed in 1991 by Microsoft. Derived from BASIC, Visual Basic is a family of general-purpose and high level programming language."[4]

Versions:

- Visual Basic 6
- Visual Basic.Net
- Visual Basic for Applications (VBA)

Syntaxes: Creating Button with Click Handler

Visual Basic 6:

```
Private Sub Command1_Click()
MsgBox "Hello, World"
End Sub
```

Visual Basic for Applications:

```
Private Sub Button1_Click()
MsgBox ""Hello, World"
End Sub
```

Visual Basic .Net:

```
Private Sub Button1_Click(sender As object,e As Even-
tArgs)Handles Button1.Click
MsgBox("Hello, World")
End Sub
```

OOP Features:

```
Dim myCharReading As New CharReader
Dim myStrReading As New StringReader
RunReader myCharReading
RunReader myStrReading
Public Sub RunReader (ByVal Reader As TextReader)
Reader.Read()
End Sub
```

Code Example: Polymorphism.

```
Public MustInherit Class TextReader
Public MustOverride Sub Read()
End Class
Public Class CharReader
Inherits TextReader
Public Overrides Sub Read()
'Code here to run this reader
End Sub
End Class
Public Class StringReader
Inherits TextReader
Public Overrides Sub Read()
'Write Code here to run this reader
End Sub

End Class
```

Code Example: Abstraction and Inheritance.

```
Public Class CharReader
Inherits TextReader
Private theDataStr As String
Public Overrides Sub Read()
'Please Write here Code here to run this reader
'Saves data to path specfied in 'theDataStr'
End Sub
Public Property DataPath() As String
Get
Return theDataStr
End Get
Set(ByVal myValue As String)
theDataStr = myValue
End Set
End Property
End Class
```

Code Example: Encapsulation.

```
Public Class GenericCollection( Of animalType)
Private theCol As New Collection
Public Sub Add( ByVal animal As animalType)
theCol.Add(animal)
End Sub
Default Public ReadOnly Property animal( ByVal Index As
Integer) As animalType
Get
Return DirectCast(theCol(Index), animalType)
End Get
End Property
End Class
```

Code Example: Generic Collection.

Uniqueness:

- **Colon (:) or End Of Line (EOL) Terminations [2]**
- **IIF (expr, truepart, falsepart)**
- **# Const Directive: It is used to define conditional compiler (Example:** #Const fruit = “Mango”**)**
- **# if...Then...#Else Directives**

Hochschule für Technik Stuttgart

References:

[1] Visual Basic, https://en.wikipedia.org/wiki/VisualBasic. Wikipedia, the freeencyclopedia. [Online; accessed 24-April-2021].

[2] Visual Basic.NET, https://en.wikipedia.org/wiki/VisualBasic.NET. Wikipedia, the freeencyclopedia. [Online; accessed 24-April-2021].

[3] Visual Basic for Applications, https://en.wikipedia.org/wiki/VisualBasicforApplications. Wikipedia, the freeencyclopedia. [Online; accessed 24-April 2021].

[4] Robert, Harkness. Malclm, Crook. David,Poves. (2007), Programming Review of Visual Basic.NET for the Laboratory Automation Industry. Industry. JALA: Journal of the Association for Laboratory Automation.

[5] White, Garry (2006). Visual Basic Programming Impact on Cognitive Development of College Students. Journal of Information Systems Education: Vol. 17 : Iss. 4 , 421-428.

[6] Introducing Generics in the CLR, issues/06/00/NET/default.aspx. Accessed May 2006.

[7] Type System, https://en.wikipedia.org/wiki/Type system Accessed April 2021.

[8] IIF, https://en.wikipedia.org/wiki/IIf Accessed April 2021.

[9] Memory leak, https://en.wikipedia.org/wiki/Memory leak Accessed April 2021.

[10] #If...Then...#Else Directives, dotnet/visual-basic/language-reference/directives/if-then-else directives Accessed May 2021.

Thank you

Concept of Programming Language : R Programming

Hochschule für Technik Stuttgart

University of Applied Sciences

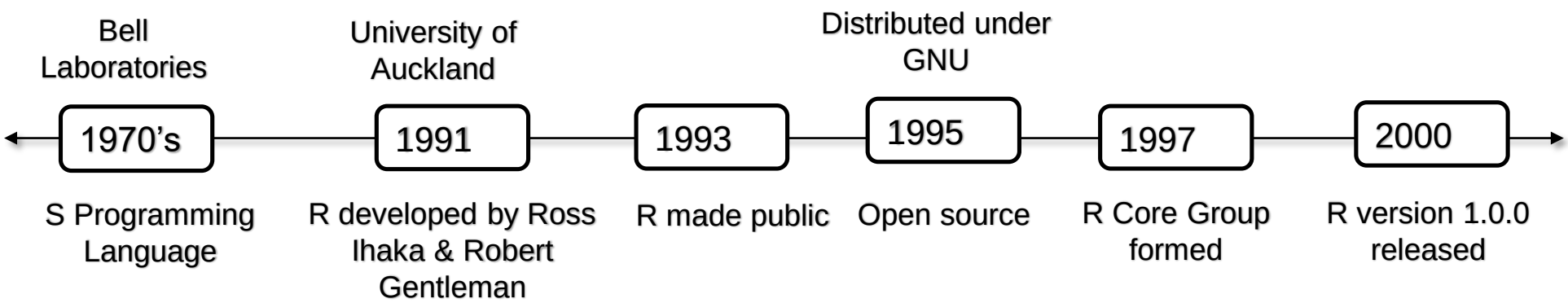
By: Joy Bharati Geetanjali

SS 2021

25th May, 2020

Evolution of R Programming

- R is a dynamic language for statistical computing and data analysis.
- Developed by Ross Ihaka and Robert Gentleman at the University of Auckland in New Zealand.
- The last stable release version is 4.1.0



FEATURES OF R

- R supports:
  - both functional programming and object oriented features.
  - mathematical features like data frames, Lists, vectors and matrices
  - extensive packages and large number of built-in formulae for Statistical modelling, graphical data visualization.

- It is an interpreted language and allows for integration of procedures written in languages such as C, C++, .Net, Python, and FORTRAN etc.

- It is platform independent and supports operating systems, like Linux, Windows, and Mac

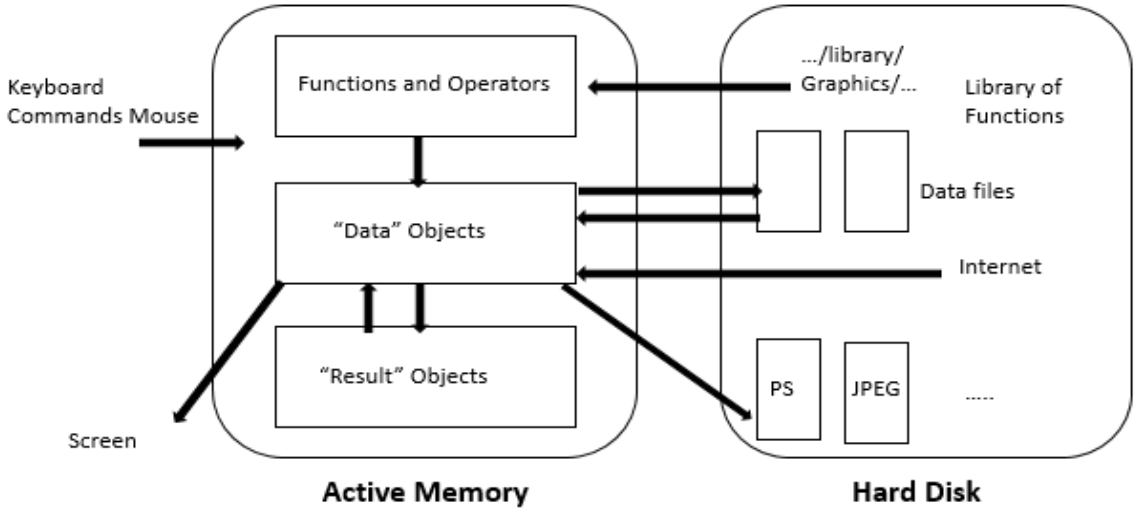
- R has its own documentation format which uses custom syntax called Rd.

- **.**Rd files can be rendered to HTML, plain text and pdf for viewing.

- It read and write in various files formats like csv, excel, xml and relational databases like MySql, Oracle, Sql server etc.

Architecture of R

- All variables, data, functions are treated as Objects
- All actions of R done on objects stored in active memory
- User functions and packages are stored in R HOME/library



Schematic view of how R works

Source: *A schematic view of how R works*, 2005, R for Beginner, by Paradis, Emmanuel, cran.r-project, 2005, pg.4

Data Types in R

- Unlike other object oriented programming languages, in R programming variables are not declared as data types, rather it gets the data type of the R - object assigned to it

- Vector: The basic data type is the R-object called vector

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

- There are 6 data types or classes of atomic vectors:

Data Type	Example	Verify in R Console	Result
Numeric	10,20,30,40.5	x <- 40.5 print (class(x))	"numeric"
Logical	TRUE, FALSE	x <- TRUE print (class(x))	"logical"
Integer	10L,20L, 30L	x <- 20L print (class(x))	"integer"
Complex	10+20i	x <- 10+20i print (class(x))	"complex"
Character	Corona', "R Program","10"	x <- 'Corona' print (class(x))	"character"
Raw	It stores "Corona" as "43 6f 72 6f 6e 61"	x <- charToRaw('Corona') print (class(x))	"raw"

Data Types in R

- Most frequently used R Objects are :
  - Vectors :

```
> fruits = c ("apple","orange","banana")
> print (class(fruits))
[1] "character"
> print (fruits)
[1] "apple"  "orange" "banana"
>
```

  - List : Contain elements and objects with different data types (for eg. Matrices, an Array and another list inside it.

    Syntax: list1 <- list(c(2,5,3),21.3,cpl)

  - Matrices : 2- dimensional set of data elements;

```
> X = matrix (c(10, 20, 30, 40,50, 60, 70, 80,90),3,3, TRUE)
> print (X)
     [,1] [,2] [,3]
[1,]   10   20   30
[2,]   40   50   60
[3,]   70   80   90
```

Source: Image made by Author, R IDE

Data Types in R

- Arrays: n- dimensional data set of same data type.
- Factors: variables that are used to store predefined number of different values i.e., categorical variables as levels. Syntax : variable <- factor(vector)

```
> factorA = factor (c(1,2,3,4,5,1,2,3,4,5))
> print (factorA)
 [1] 1 2 3 4 5 1 2 3 4 5
Levels: 1 2 3 4 5
>
```

- Data Frames: unlike an Array, data frame is a 2 dimensional data structure, that allows different data types (values of data tables)

```
> dataA = data.frame (values = c (2019,2020,2021), diesease =
 c('Influenza','Corona','NoCorona'))
> print (dataA)
  values  diesease
1   2019 Influenza
2   2020    Corona
3   2021  NoCorona
>
```

Source: Image made by Author, R IDE

Operators in R

- 4 main categories of Operators in R

  - Arithmetic Operator (+,-,*,/,%%;%/%,ˆ)
  - Relational Operator (>,<,==,<=,>=,!=)
  - Logical Operators (&,,!,&&,|,||)
  - Assignment Operators (=,<-,->,<<-,->>)

- Miscellaneous Operators in R programming

Operator	Description	Usage
:	Creates series of numbers from left operand to right operand	a:b
%in%	Identifies if an element(a) belongs to a vector(b)	a %in% b
%*%	Performs multiplication of a vector with its transpose	A %*% t(A)

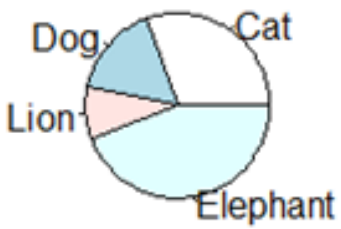
```
> y = matrix(c(10,20,30,40,50,60,70,80,90),3,3, TRUE)
> print (y %*% t(y))
     [,1]  [,2]  [,3]
[1,] 1400  3200  5000
[2,] 3200  7700 12200
[3,] 5000 12200 19400
>
> x = c (10,20,30)
> y = 20
> print (y %in% x)
[1] TRUE
>
> x = 10:20
> print (x)
 [1] 10 11 12 13 14 15 16 17 18 19 20
>
```

Source: Image made by Author, R IDE

CHARTS AND GRAPHS

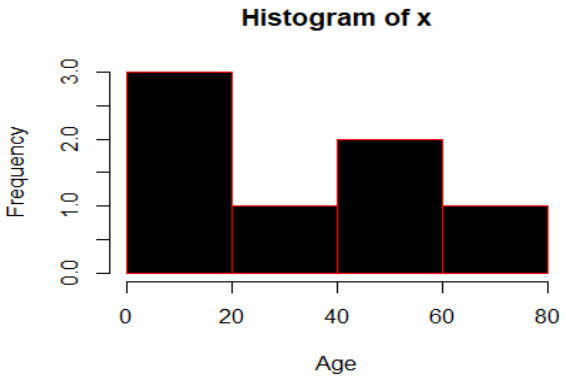
- Extensive ready to use functions for Statistical analysis, data visualization and graphical interpretations using R
- Two types of graphical functions: the high-level plotting function to create a new graph and the low-level plotting function to add elements to an existing graph.
- In R every input and output as treated as an object but for graphical functions, the output can not be assigned to an object sent to a graphical device or file.
- Examples:

```
> x<- c (66,35,20,95)
> labels <- c ("Cat","Dog","Lion","Elephant")
> pie(x,labels)
```



Example of Pie-chart in R IDE

```
> x <- c(10,45,5,50,15,25,70)
> hist (x,xlab = "Age", col = "Black", border = "red")
>
```



Example of Histogram in R IDE

Source: Image made by Author, R IDE

Disadvantages of R

- **Proper coding standards are not maintained**
- **Performance issues**: Not as fast as compare to its competitor programming languages, especially when displaying image of higher resolution or working with large matrices
- **Memory management issues** especially when processing large datasets. The execution environment of R is unrestricted, which could result in malicious behavior or excessive use of hardware resources in a shared environment.
- **Security concerns**: As full audit of contributed code is not feasible by Comprehensive R archive network (CRAN) maintainers, lacks security features.

Conclusion

- R provides an interactive platform for statistical modelling, data analysis and visualization
- Widely used in research and by academic users for its usability and flexibility
- Extensive library and continuously growing packages
- CRAN team and large community support for knowledge sharing among R users.

References

- Hadley Wickham, Advanced R ISBN-13: 978-1466586963
- Emmanuel Paradis, R for Beginner URL: https://cran.r-project.org/ doc/contrib/Paradisrdebuts en.pdf
- R. Ihaka, R. Gentleman, R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics, 5(3):299314, 1996
- Colin Gillespie, Robin Lovelace, Efficient R Programming
- Roger D. Peng, R Programming for Data Science
- Thomas H. Davenport, How Analytics Has Changed in the Last 10 Years (and How Its Stayed the Same).In Harvard Business Review
- The Comprehensive R Archive Network URL: http://cran.r-project. org.
- F. Morandat, B. Hill, L. Osvald, and J.Vitek. Evaluating the Design of the R Language. In: ECOOP12.
- R Language Definition. R Development Core Team. 2010. ISBN: 3- 900051-13-5.
- Ooms, J. (2013). The R app ArmorPackage: Enforcing Security Policies in R Using Dynamic Sandboxing on Linux. Journal of Statistical Software, 55(7). https://doi.org/10.18637/jss.v055.i07

Questions? Questions!

THANK YOU VERY MUCH FOR YOUR ATTENTION!

APPENDIX

OOPs in R

Different object-oriented systems or object models in R :

- **S3:** It implements generic function and message passing OO (object-oriented). With message passing, messages are sent to objects and the objects determines which functions to call.

- **S4:** It has formal class definitions, which describes the representations and Inheritance for each class. It also has special helper function for defining generic function and methods.

- **Reference Class (RC)**: It implements message passing O-O concepts, so methods belong to classes and not functions.

Performance of R

- **Code Optimization :**
  - Using matrix instead of data frame; splitting bigger data objects to smaller ones; changing the type and size of an object in R should be avoided

- **Using RProf() :** It is a function included in the base package utils and can be used to find slow spots in a. Profiling is optional and can also be turned off

- **Byte Code Compilation** : Compilation is handled in two ways:
  - it can either be called explicitly by calling a certain function to carry out the compilation, or
  - it can be called implicitly by enabling compilation to occur automatically at certain points.

  Using Just-In-Time and arithmetic compiler package enables the code to run much faster.