Sejam bem-vindos!

Ciência da Computação

Paradigmas de linguagem de programação

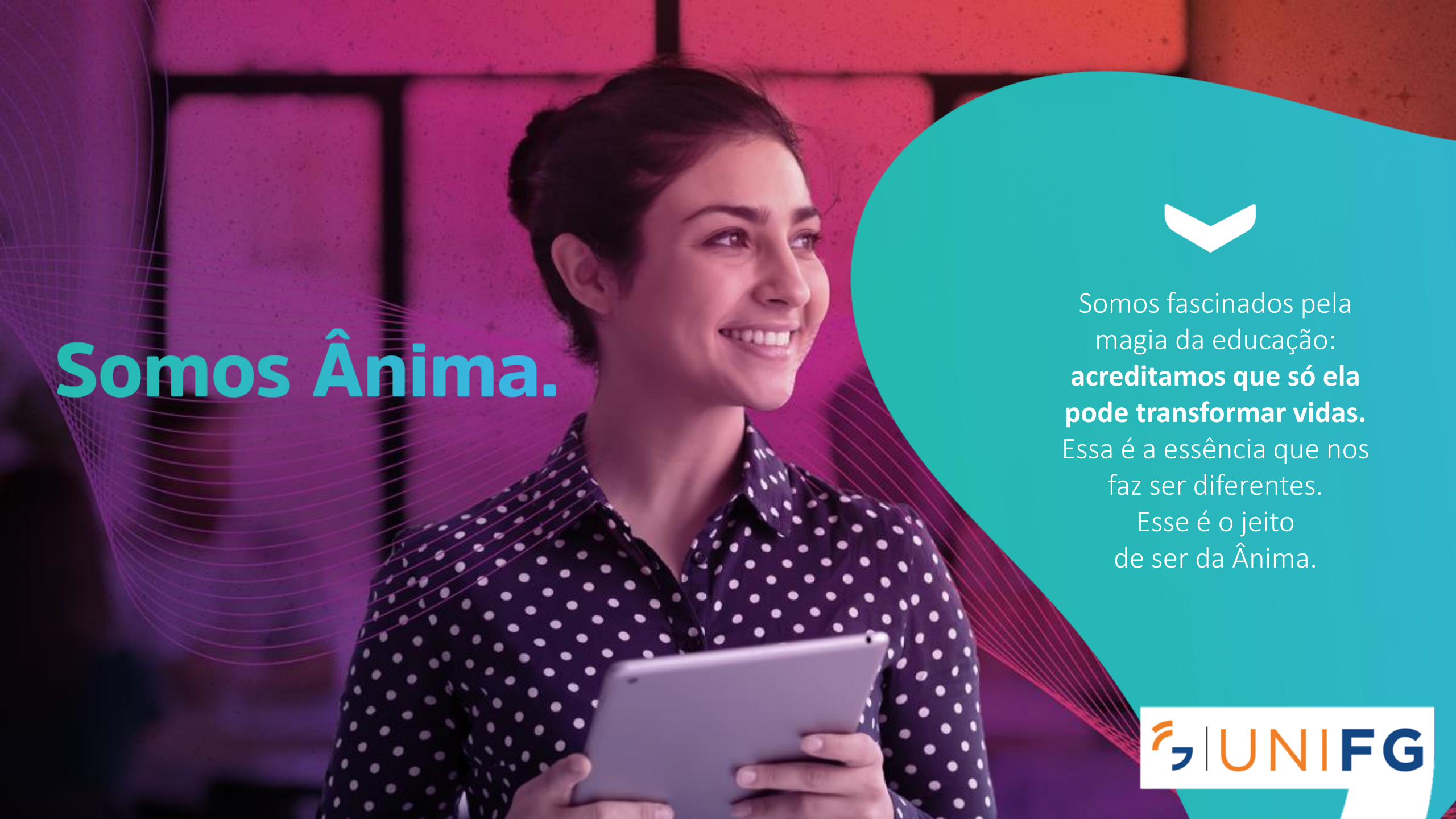
Diogenes Carvalho Matias - diogenes.matias@unifg.edu.br

2021.2

TRANS_
FORMAR
O PAÍS PELA
EDUCAÇÃO
É O QUE
NOS MOVE

ecossistema ânima

UNIFG



Somos fascinados pela
magia da educação:
acreditamos que só ela
pode transformar vidas.
Essa é a essência que nos
faz ser diferentes.
Esse é o jeito
de ser da Ânima.

UNIFG

Quem sou

Quem sou

Nosso cronograma...

Nossas **AVALIAÇÕES**

Nota	Atividade	Descrição	Data
N1	A1	Práticas sobres o conteúdo	Até 23.10.2021
N2	APS	Atividade Prática Supervisionada	18 a 22.10.2021
	A2	Prova no BB – Objetiva e Subjetiva	09 a 15.12.2021

Paradigmas de linguagem de programação

UNIFG

Introdução

Uma boa linguagem de programação é um universo conceitual para pensar em programação.

A. Perlis

Introdução

As linguagens de programação têm quatro propriedades:

- Sintaxe
- Nomes
- Tipos
- Semântica

Para qualquer idioma:

Seus projetistas devem definir essas propriedades;
Seus programadores devem dominar essas propriedades.

Syntax

A sintaxe de uma linguagem de programação é uma descrição precisa de todos os seus programas gramaticalmente corretos.

Ao estudar a sintaxe, fazemos perguntas como:

Qual é a gramática para a língua?
Qual é o vocabulário básico?
Como os erros de sintaxe são detectados?

Nomes

Vários tipos de entidades em um programa têm nomes:

variáveis, tipos, funções, parâmetros, classes, objetos, ...

As entidades nomeadas estão vinculadas em um programa em execução para:

Escopo;
Visibilidade;
Tipo;
Tempo de vida;

Tipos

Um tipo é uma coleção de valores e uma coleção de operações sobre esses valores.
Tipos simples:
números, caracteres, booleanos, ...

Tipos estruturados:
Strings, listas, árvores, tabela hash, ...

O sistema de tipo de um idioma pode ajudar a:
Determinar operações legais;
Detectar erros de tipo;

Semanticas

O significado de um programa é chamado de semântica.

Ao estudar a semântica, fazemos perguntas como:

Quando um programa está em execução, o que acontece com os valores das variáveis?
O que significa cada declaração?
Que modelo subjacente rege o comportamento de tempo de execução, como a chamada de função?
Como os objetos são alocados na memória no tempo de execução?

Paradigma Imperativo

Segue o modelo clássico von Neumann-Eckert:
Programa e dados são indistinguíveis na memória
Programa = uma sequência de comandos
Estado = valores de todas as variáveis quando o programa é executado
Grandes programas usam abstração processual

Exemplos de idiomas imperativos:
Cobol, Fortran, C, Ada, Perl, ...

O modelo von Neumann-Eckert

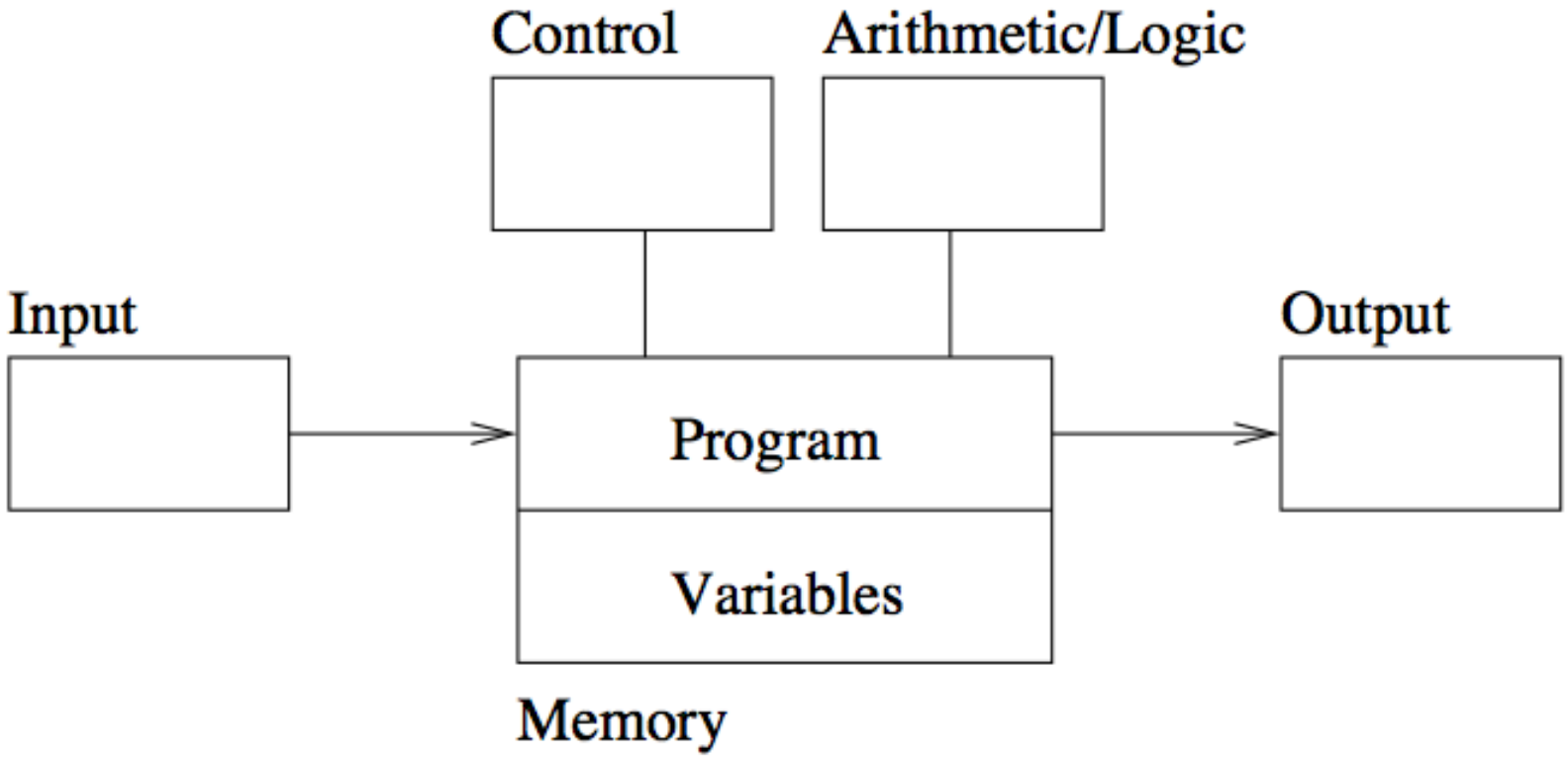


Figure 1.1: The von Neumann-Eckert Computer Model

Paradigma orientado a objetos (OO)

Um Programa OO é uma coleção de objetos que interagem passando mensagens que transformam o estado.
Ao estudar OO, aprendemos sobre:
Enviando Mensagens
Herança
Polimorfismo

Exemplos de idiomas OO:

Smalltalk, Java, C++, C# e Python

Paradigma Funcional

A programação funcional modela uma computação como uma coleção de funções matemáticas.
Entrada = domínio
Saída = alcance

As linguagens funcionais são caracterizadas por:
Composição funcional
Recursão
Exemplos de linguagens funcionais:
Lisp, Scheme, ML, Haskell, ...

Paradigma Lógico

A programação lógica declara que resultado o programa deve realizar, e não como deve ser realizado.

Ao estudar programação lógica vemos:

Programas como conjuntos de restrições sobre um problema;
Programas que alcançam todas as soluções possíveis;
Programas não deterministas;

Exemplos de linguagens de programação lógica:
Prólogo

Tópicos Especiais

Manipulação de eventos
Por exemplo, GUIs, sistemas de segurança doméstica

Simultaneidade
Por exemplo, programas de servidor de clientes

Exatidão
Como podemos provar que um programa faz o que deve fazer sob todas as circunstâncias?
Por que isso é importante???

Uma Breve História

Como e quando as linguagens de programação evoluíram?
Que comunidades as desenvolveram e usaram?

- Inteligência artificial;
- Educação em Ciência da Computação;
- Ciência e Engenharia;
- Sistemas de Informação;
- Sistemas e Redes;
- World Wide Web;

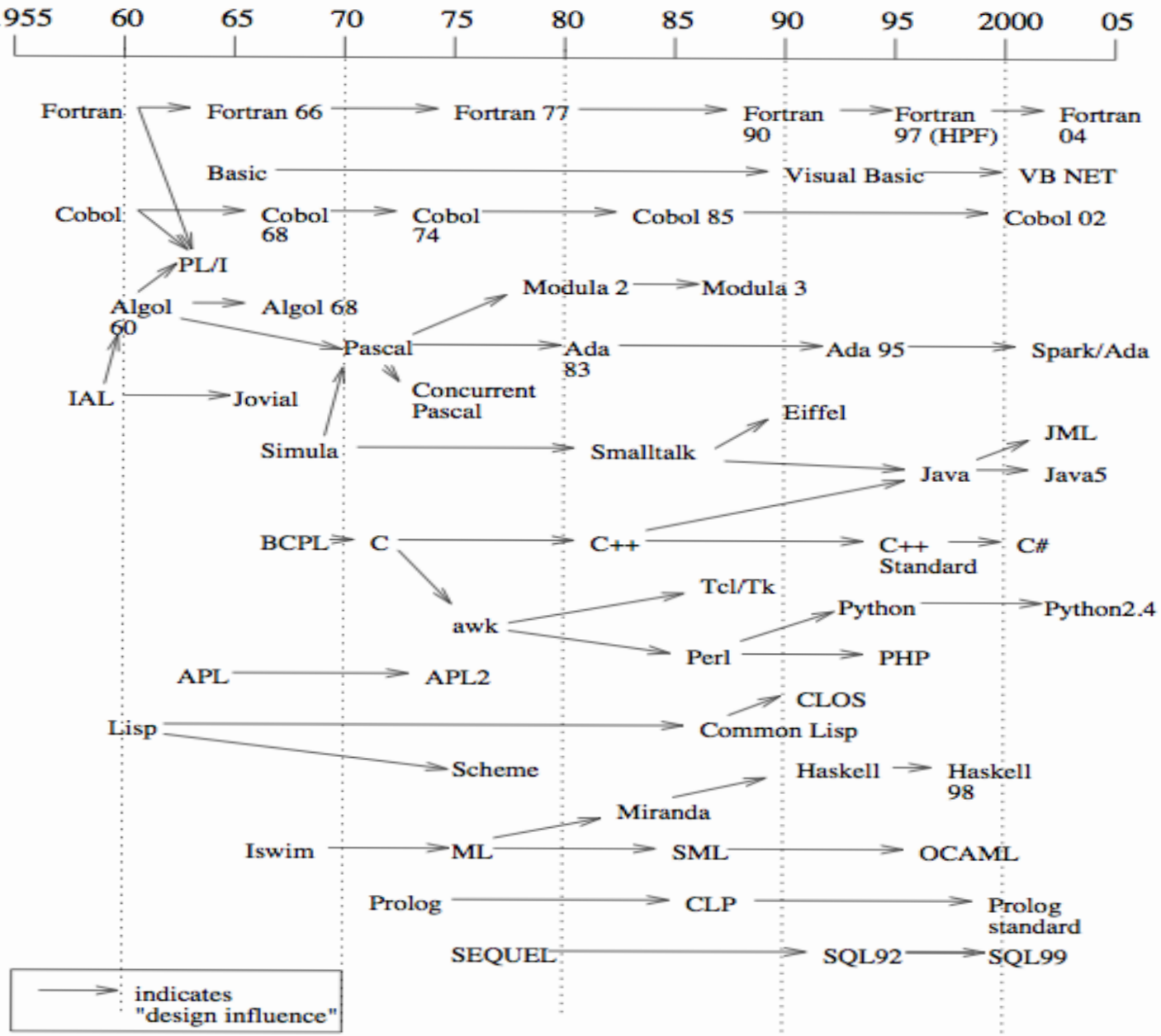


Figure 1.2: A Snapshot of Programming Language History

Design de linguagem

Restrições de design
- Arquitetura de computador;
- Configuração técnica;
- Padrões;
- Sistemas legados;

Resultados e Metas de Design

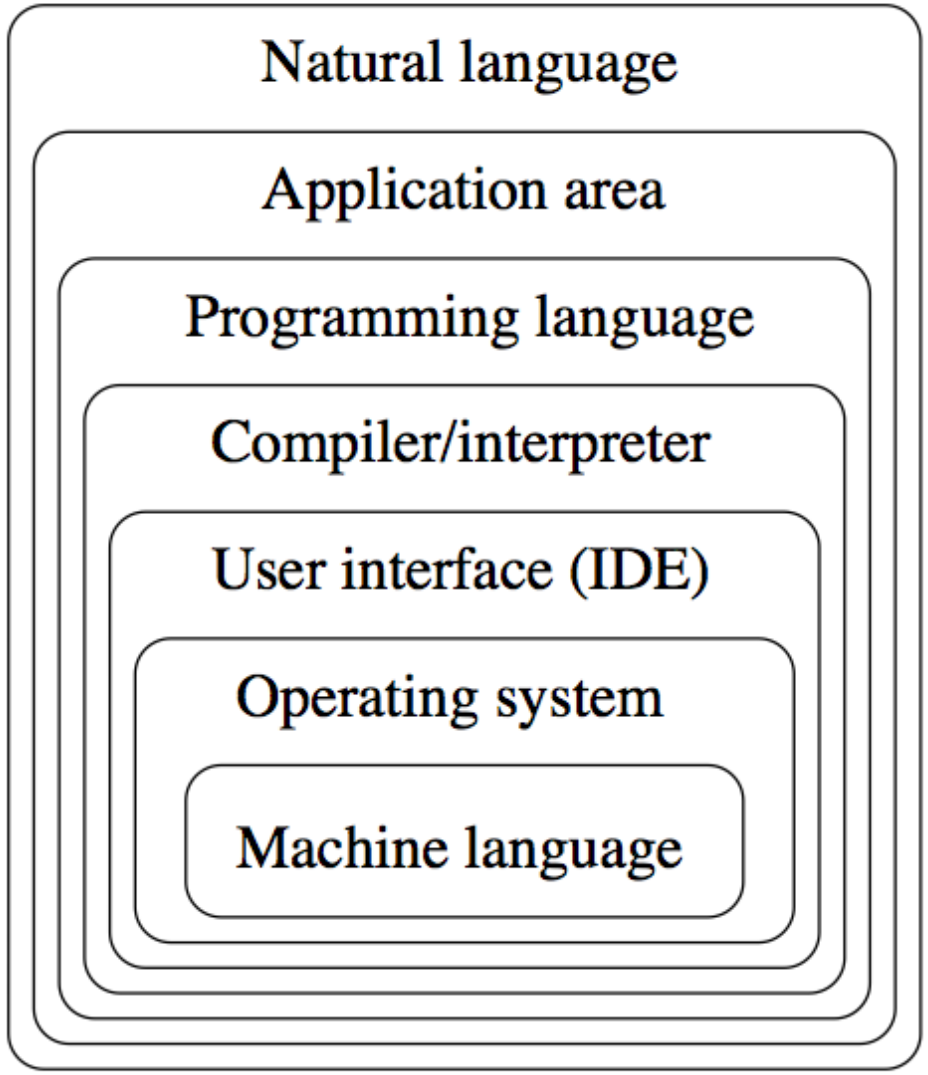


Figure 1.3: Levels of Abstraction in Computing

Design de linguagem

Restrições de design
- Arquitetura de computador;
- Configuração técnica;
- Padrões;
- Sistemas legados;

Resultados e Metas de Design

O que faz uma linguagem de sucesso?

Características-chave:

- Simplicidade e legibilidade
- Clareza
- Fiabilidade
- Suporte
- Abstração
- Implementação eficiente

Simplicidade e Legibilidade

- Pequeno conjunto de instruções
  - Por exemplo, Java vs Scheme
- Sintaxe simples
  - Por exemplo, C/C++/Java vs Python
- Benefícios:
  - Facilidade de aprendizado
  - Facilidade de programação

Clareza sobre a Vinculação

Um elemento de linguagem está vinculado a uma propriedade no momento em que a propriedade é definida para ela.

Assim, uma vinculação é a associação entre um objeto e uma propriedade desse objeto

Exemplos:

- uma variável e seu tipo;
- uma variável e seu valor;
- A vinculação antecipada ocorre no tempo de compilação;
- A vinculação tardia ocorre no tempo de execução;

Fiabilidade

Uma linguagem é confiável se:

- O comportamento do programa é o mesmo em diferentes plataformas
  - Por exemplo, versões iniciais do Fortran
- Erros de tipo são detectados
  - Por exemplo, C vs Haskell
- Erros semânticos estão devidamente presos
  - Por exemplo, C vs.C++
- Vazamentos de memória são evitados
  - Por exemplo, C vs Java

Suporte ao idioma

Compiladores/intérpretes acessíveis (domínio público)
- Bons textos e tutoriais
- Ampla comunidade de usuários
- Integrado com ambientes de desenvolvimento (IDEs)

Abstração em Programação

Dados
- Tipos/classes definidos pelo programador
- Bibliotecas de classe

Processual
- Funções definidas pelo programador
- Bibliotecas de funções padrão

Ortogonal

Uma língua é ortogonal se suas características são construídas sobre um pequeno conjunto mutuamente independente de operações primitivas.

Menos regras excepcionais = simplicidade conceitual

Por exemplo, restringir tipos de argumentos a uma função

Implementação eficiente

- Sistemas embarcados
  - Capacidade de resposta em tempo real (por exemplo, navegação)
  - Falhas das implementações iniciais de Ada
- Aplicações web
  - Resposta aos usuários (por exemplo, pesquisa no Google)
- Aplicativos de banco de dados corporativos
  - Pesquisa e atualização eficientes
- Aplicações de IA
  - Modelando comportamentos humanos

Compiladores e Máquinas Virtuais

Compilador – produz código de máquina
Intérprete – executa instruções em uma máquina virtual
Exemplos de idiomas compilados:
    Fortran, Cobol, C, C++
Exemplos de idiomas interpretados:
    Esquema, Haskell, Python
Compilação/interpretação híbrida
    A Máquina Virtual Java (JVM)

O processo de compilação

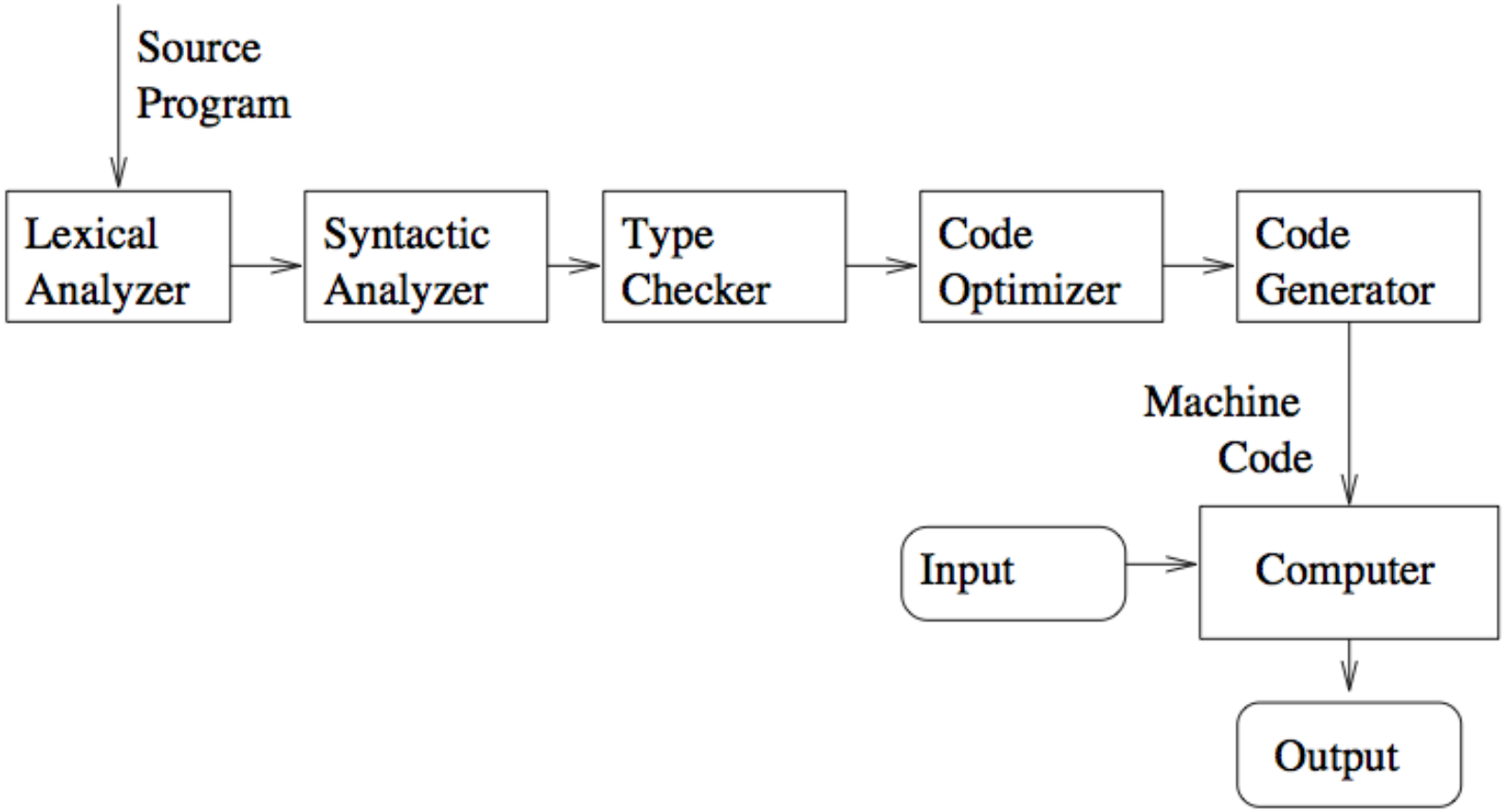


Figure 1.4: The Compile-and-Run Process

O Processo de Interpretação

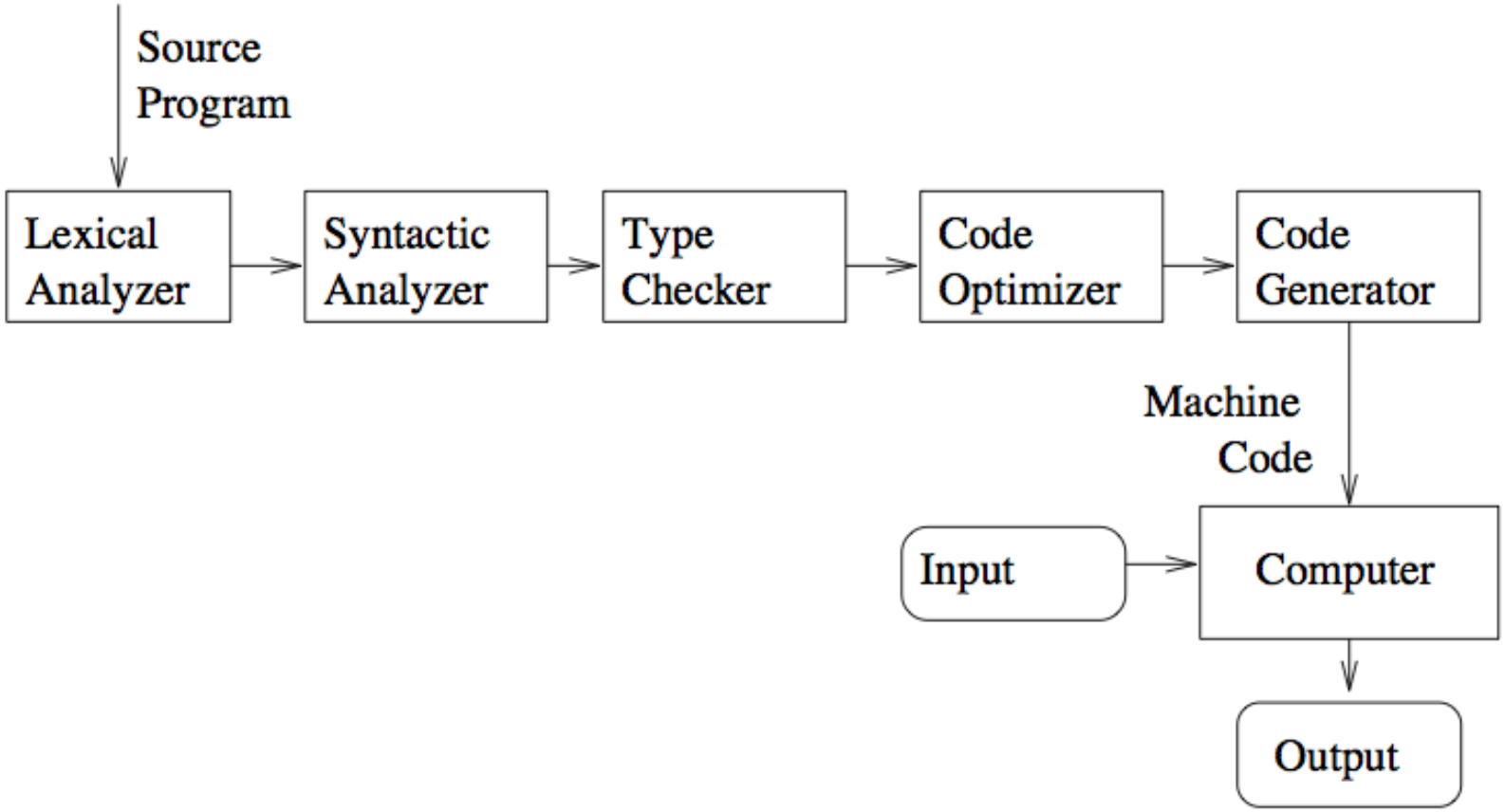


Figure 1.4: The Compile-and-Run Process

Exercício para casa em casa ☺

1.Comente a seguinte citação:

É praticamente impossível ensinar boa programação aos alunos
que tiveram uma exposição prévia ao BASIC; como programadores potenciais
eles são mentalmente mutilado além da esperança de regeneração. – E. Dijkstra

2-Dê uma instrução de exemplo em sua língua favorita que é particularmente ilegível.
Por exemplo, o que significa a expressão C (*p++ = *q++) significa?

Fique

Ligado

“Educação não transforma o mundo. **Educação** muda as pessoas. **Pessoas** transformam o **mundo**.”

Paulo Freire

Bom semestre!

CAMPI

PIEDADE - Rua Comendador José Didier, 27
Piedade, Jaboatão dos Guararapes - PE

BOA VISTA - Av. Gov. Carlos de Lima Cavalcanti, 110,
Derby, Recife - PE

UNIFG