Algoritmos e Programação I: Roteiro Aula 01*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
https://ava.ufms.br

Sumário

1 Computação

A primeira pergunta que um aluno de computação deve saber responder é: **o que é computação?**

Computação pode ser definida como:

Definição 1 *Computação é o estudo de processos sistemáticos que descrevem e transformam informação: sua teoria, análise, projeto, eficiência, implementação e aplicação. A questão fundamental relativa a toda computação é: O que pode e o que não pode ser automatizado.*

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas constantes na página da disciplina

Computação inclui o projeto e a construção de sistemas de hardware e software para um amplo escopo de aplicações; processamento, estruturação, e gerenciamento de vários tipos de informação; elaboração de estudos científicos usando computadores; introduzindo inteligência em sistemas computacionais; criando e usando comunicação e mídia para entretenimento; procura e recuperação de informações relevantes, e por ai adiante. A lista é virtualmente sem fim, e as possibilidades amplas.

Abordaremos a computação identificando suas nove áreas principais além da sua dimensão social e profissional. Quatro destas áreas - algoritmos e estrutura de dados, arquitetura, linguagens de programação e metodologia de software - serão enfatizadas nesta disciplina.

2 Uma Breve História da Computação

Existem dois conceitos importantes na história da computação: a **mecanização da aritmética** e o conceito de **programa armazenado** para o controle automático de computações.

Leia atentamente as transparências da Aula Teórica 00, visitando todos os hiperlinks.

3 Teoria, Abstração, Projeto e Contexto Social

A computação pode ser vista como compreendendo três paradigmas (modelos) distintos, ou modos de abordagem, os quais são denominados **teoria**, **abstração** e **projeto**. Além disso, todos os três paradigmas existem com um **contexto social** muito bem definido. Isto é, a computação e os produtos que a circundam existem com o propósito de fornecer aos sistemas internacionais, de comércio, de governo, científicos, e sociais um funcionamento efetivo e a realizar tarefas que de outra forma não poderiam ser efetuadas. Cientistas da computação, mais que os profissionais de outras ciências, são influenciados em maior intensidade pelo contexto social relacionado ao seu trabalho, visto que a computação nos dias atuais está inserida em todas as áreas do conhecimento e o seu trabalho será o de automatizar processos que não estão sendo realizados de forma satisfatória por seres humanos.

Uma pessoa que trabalha com o paradigma de teoria tende a ter um ponto de vista matemático, e portanto utilizará definições formais, axiomas, teoremas e demonstrações como um meio de explorar suas ideias em computação. O processo de abstração é fundamentado nas ciências. Como o estudo de física, química e biologia, o ponto de vista científico da computação está relacionado com a formulação de hipóteses, construção de modelos, estabelecimento de previsões, executando experimentos e testando os resultados. Projeto tem sua fundamentação em engenharia. O ponto de vista da engenharia com relação à computação está relacionado com a factibilidade do sistema, custos, eficiência e alternativas. O profissional de computação orientado para engenharia estará interessado em atividades como estimativa de requisitos, formulação de especificações, preparação e implantação de um projeto, e teste e avaliação de um projeto.

Estes três paradigmas são complementares e interdependentes quando usados em computação. Isto é, a solução da maioria dos problemas computacionais requer alguma combinação de teoria, abstração e projeto. Desta forma cientistas da computação e engenheiros de computação estarão melhores preparados se eles desenvolverem o interesse por todos os três paradigmas, mesmo que eles possam se especializar em somente um ou dois.

Analisando o que foi visto anteriormente (Uma Breve História da Computação), vimos exemplos de todos os três modos de pensamento. Os antigos Gregos usaram teoria (método axiomático e o estudo de padrões formais de raciocínio), abstração quando eles estudaram o raciocínio, observaram padrões nele, e formularam outros padrões. Ao contrário, os Egípcios

e Babilônios usaram abstração e projeto ao invés de teoria. Eles usaram abstração na formulação de problemas utilizando uma forma padrão, tais como problemas de cálculo de juros, e eles usaram projeto no desenvolvimento de métodos para solucionar tais problemas.

A introdução da álgebra por Viète envolveu abstração (traduzindo problemas para uma forma simbólica); a invenção dos logaritmos por Napier envolveu teoria; e a invenção da régua de cálculo envolveu projeto. O trabalho de Galileu envolveu tanto abstração como teoria - abstração no fato de que ele usou ferramentas matemáticas para modelar eventos e processos no mundo físico a obter dados para testar os modelos e teoria à medida que ele usou técnicas matemáticas para trabalhar com esses modelos. Os esforços de Pascal e Schickard na tentativa de construir os primeiros dispositivos computacionais e o trabalho de Babbage são exemplos do sucesso da atividade de projeto. O trabalho de Boole é um ótimo exemplo tanto de teoria como de abstração. Na tentativa de modelar o raciocínio usando símbolos, ele estava usando abstração, e no trabalho com o modelo para deduzir conclusões, ele estava usando teoria. Quevedo, Hollerith, Eckert, Aiken e Atanasoff todos fizeram importantes contribuições para o projeto. Hilbert, Gödel e Turing fizeram importantes contribuições para teoria.

É importante notar que duas figuras importantes da história da computação - Leibniz e von Neumann - utilizaram, de maneira significativa, todos os três modos de pensamento. Leibniz utilizou a teoria no estudo dos números binários. Sua visão de reduzir o raciocínio a um conjunto de operações foi uma tentativa de modelar simbolicamente o raciocínio, e desta forma ele usou abstração. E seu desenvolvimento da roda de Leibniz é um exemplo de projeto. De forma semelhante, o conhecimento e utilização do trabalho de Gödel e Turing por von Newmann contribuiu para a teoria. A formulação dos conceitos de programa armazenado e operação sequencial do processador central são exemplos de abstração. E a utilização destas noções na proposta detalhada para o conjunto de instruções e os circuitos lógicos do EDVAC são exemplos de projeto.

Na disciplina de Algoritmos e Programação I exploraremos todos os três métodos de pensamento. Nosso objetivo é de proporcionar um repertório de ferramentas intelectuais que proporcione ao estudante compreender e solucionar uma variedade de problemas relacionados com computação. Além disso introduziremos uma metodologia para projeto de algoritmos, Método para Solução de Problemas Algorítmicos (MSPA).

4 As Nove Áreas da Computação

O processo de teoria, abstração e projeto, em conjunto com o contexto social, influencia o trabalho em nove diferentes áreas principais que compõem a disciplina de computação.

A Figura 1 ilustra as nove áreas da Computação. Na área oval interna (em amarelo) da figura estão as áreas consideradas fundamentais. Na disciplina de Algoritmos e Programação I daremos início ao estudo de tópicos relacionados na área amarela da figura 1. Esses tópicos são fundamentais para o início do estudo da computação.

- **Algoritmo e Estrutura de Dados** - Modelos e metodologias específicas são utilizados para representar a informação e solucionar problemas computacionais. No projeto e implementação dos algoritmos na disciplina, utilizaremos uma metodologia específica para solucionar problemas algorítmicos, além disso, utilizaremos estruturas de dados simples, vetores, matrizes, listas e estruturas.

- **Linguagens de Programação** - Para representar as soluções de problemas algorítmicos, programadores usam um estilo "artificial" de linguagem que pode ser entendida tanto por humanos quanto ser interpretada por computadores. Nesta disciplina, só utilizaremos uma linguagem de programação para representar as soluções algorítmicas.

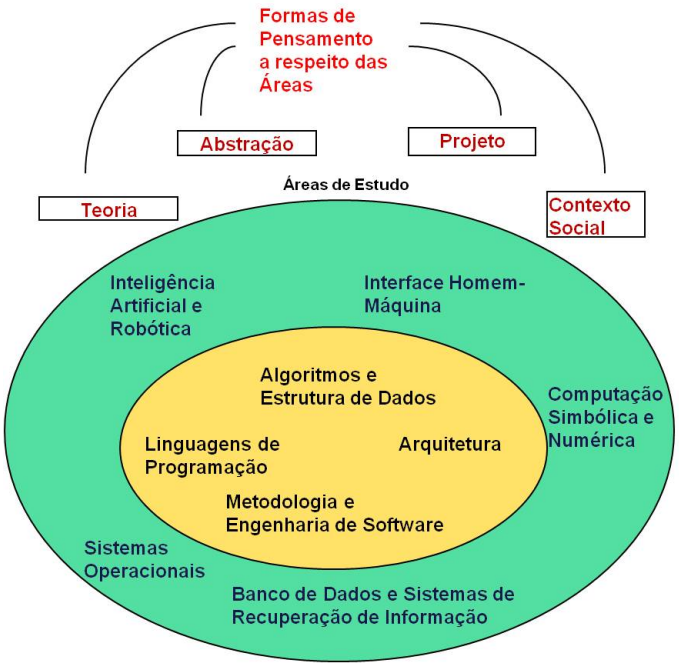

Figura 1: As Nove Áreas da Computação - Allen B. Tucker et al.

Discutiremos as implementações dos algoritmos usando a linguagem Python. A área de Linguagens de Programação envolve o projeto e a implementação de uma linguagem.

- **Arquitetura** - Necessitamos aprender como os computadores são organizados e projetados desta forma eles podem executar de forma eficiente e correta os passos de um programa. A máquina de von Neumann é uma das arquiteturas iniciais. Na nossa disciplina iremos apenas apresentar os conceitos necessários para o entendimento de como um programa é executado num computador.

- **Metodologia e Engenharia de Software** - Programadores necessitam métodos projetados para criação efetiva de sistemas de software seguro, eficiente e confiável para computadores. Durante a disciplina introduziremos alguns princípios básicos de metodologia de software, incluindo tanto métodos formais quanto não formais para assegurar o confiabilidade do software. Abordaremos os conceitos necessários para o desenvolvimento correto de um programa e que posteriormente serão ampliados e detalhados quando do projeto e análise de um sistema.

5 Sistemas de Computação

Uma outra pergunta que um aluno de computação deve responder é: **o que faz um profissional da área de computação?**

De uma forma simplificada, o profissional da área de computação trabalha na solução de problemas usando um sistema de computação.

Sistemas de Computação são encontrados em toda a parte. Um sistema de computação pode ser visto como um computador. Mas o que é um computador? Um computador é um sistema feito de dois componentes principais: hardware e software. O hardware do computador é um equipamento físico. O software é a coleção de programas (instruções) que possibilita que o hardware faça o seu trabalho. A Figura 2 representa um **sistema de computação**.

Na nossa discussão da história da computação, notamos que Babbage projetou sua máquina analítica como um sistema de diversos componentes separados, cada um deles com sua função específica. Este esquema geral foi incorporado posteriormente nos computadores e é, de fato, a característica comum da maioria dos computadores modernos. Vamos descrever brevemente os principais componentes de um sistema de computação moderno.

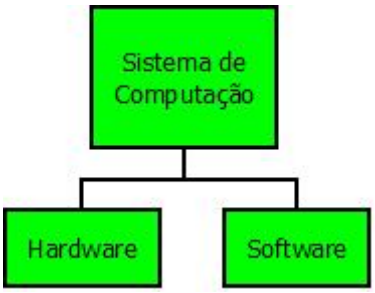

Figura 2: Um Sistema de Computação.

5.1 Hardware

A parte central de qualquer sistema de computação é sua **unidade central de processamento**, ou **CPU**. A CPU controla as operações de todo o sistema, executa operações aritméticas e lógicas, e armazena e recupera instruções e dados. As instruções e dados são armazenados em uma **unidade de memória** de alta velocidade, e a **unidade de controle** traz (*fetches*) essas instruções da memória, decodifica-as e controla o sistema para executar as operações indicadas nas instruções. As operações que são aritméticas ou lógicas são executadas usando registradores especiais e circuitos da **unidade lógico aritmética** (**ALU**) da CPU.

A unidade de memória é chamada a **memória primária** ou **principal** ou **interna** do sistema de computação. Ela é utilizada para armazenar as instruções e dados dos programas sendo executados. A maioria dos sistemas de computação também têm componentes que servem como **memória secundária** ou **auxiliar** ou **externa**. Formas comuns desse tipo de memória são memória *flash*, cartões de memória, dvds, hd externo, etc. Esses dispositivos periféricos fornecem armazenagem de longa duração para grandes coleções de informações. A taxa de transferência de/para esses dispositivos é bem menor que a da memória principal.

Outros periféricos são usados para transmitir instruções, dados e resultados entre o usuário e a CPU. Eles são os **dispositivos de entrada/saída** e podem ter diversas formas tais como teclados, vídeos, leitoras de cartões, impressoras, *scanners*, etc. Sua função é a de converter informações de uma forma externa que é inteligível pelo usuário para uma forma que possa ser processada pelo sistema de computação, e vice-versa.

A Figura 3 mostra o relacionamento entre esses componentes num sistema de computação. As setas indicam como a informação flui através do sistema.

5.1.1 Memória

Os dispositivos que compõe a unidade de memória de um computador são dispositivos de dois estados. Se um dos estados é interpretado como 0 e o outro como 1, então é natural utilizar um **esquema binário**, usando somente os dois dígitos binários (**bits**) 0 e 1 para representação de informações num computador. Esses dispositivos de dois estados são organizados em grupos chamados **bytes**, cada um dos quais contém um número fixo desses dispositivos, usualmente oito, e desta forma pode armazenar um número fixo de bits.

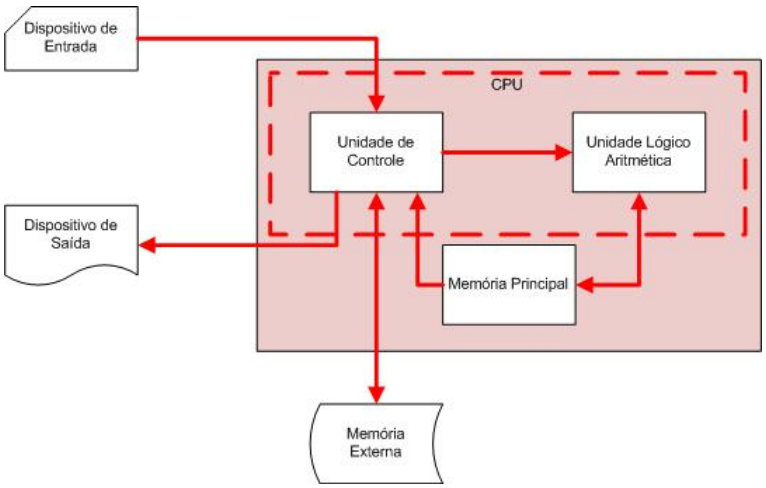

Figura 3: Principais Componentes de Hardware de um Sistema de Computação.

Usualmente, a memória é medida em bytes, 256K de memória refere-se a uma memória que consiste de $2^8 \times 2^{10} = 262.144$ bytes ($1K = 2^{10} = 1.024$), ou $2^{21} = 2.097.152$ bits.

Os bits e bytes são agrupados em **palavras**. O comprimento das palavras varia de acordo com o computador, os tamanhos usuais são 32 bits (= 4 bytes) e 64 bits (= 8 bytes). Cada palavra é identificada por um **endereço** e pode ser acessada diretamente usando este endereço. Isso torna possível armazenar informação numa palavra específica da memória e recuperá-la posteriormente.

Na disciplina de Introdução a Sistemas Digitais todos esses aspectos serão abordados em detalhes.

5.2 Software

Independente da arquitetura do hardware, o software de um sistema de computação é dividido em duas categorias principais: software do sistema e software de aplicação. Software do sistema gerencia os recursos do computador providenciando a interface entre o hardware e os usuários, mas não executa nada que resolva um problema específico do usuário. Por outro lado, o software de aplicação é responsável por auxiliar os usuários a resolver um problema com o auxílio de um computador.

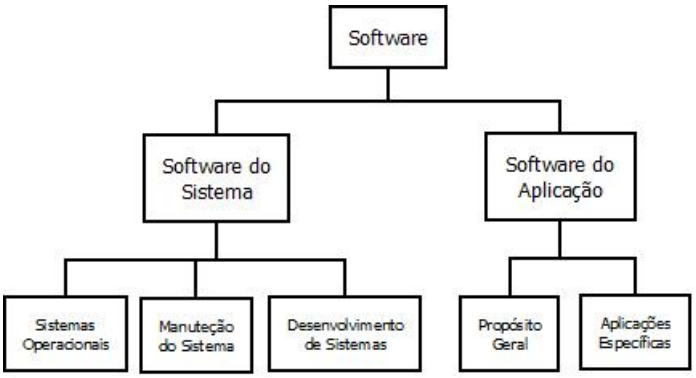

Figura 4: Tipos de Software.

O **software do sistema** é formado por um conjunto de programas que gerenciam os recursos de hardware e executam tarefas necessárias para o processamento da informação. Esses programas são divididos em três classes: sistema operacional, manutenção do sistema e desenvolvimento do sistemas.

O **sistema operacional** fornece os serviços necessários para a utilização do hardware, tais como interface do usuário, acesso a arquivos e banco de dados, e interfaces de com o sistema de comunicação tal como os protocolos de internet. O principal objetivo deste software é manter o sistema operando de forma eficiente e ao mesmo tempo permitindo que os usuários acessem o sistema. O Linux, Windows, Android e MacOS são exemplos de sistemas operacionais.

Software de manutenção do sistema fornecem ferramentas do sistema e outros serviços operacionais. Exemplos de ferramentas de serviço são programas de ordenação, formatadores de disco, etc. Serviços operacionais consistem de programas que medem o desempenho do sistema, protegem o sistema, etc.

A última categoria de software do sistema é o **software de desenvolvimento de sistemas** que inclui tradutores de linguagem que convertem programas em linguagem de máquina para execução, ferramentas de depuração de programas e ferramentas CASE.

O **software de aplicação** e dividido em duas classes: software de propósito geral e software específico de aplicação.

Software de propósito geral é distribuído pelos desenvolvedores, usando várias formas de licenciamento e pode ser utilizado para mais de uma aplicação. Exemplos de software de propósito geral incluem processadores de texto, sistemas de gerenciamento de banco de dados e sistemas de auxílio para projeto de sistemas.

Software específico de aplicação somente podem ser utilizados para seu propósito específico. Um exemplo é o SISCAD que faz o gerenciamento acadêmico da UFMS.

6 Computadores, Programas e Vida Diária

Antes de iniciarmos nosso estudo formal de computação, necessitamos dar uma olhada no mundo real; é muito importante convencer-nos de que os princípios e práticas que esta disciplina aborda estão fortemente relacionadas com a vida do dia-a-dia. A história já nos mostrou que a computação já existe de alguma forma desde os antigos Gregos. Além disso, na nossa experiência diária temos que, computadores, algoritmos, solução de problemas e programas estão onipresentes e tem efeitos fundamentais na forma como vivemos.

6.1 Por que Estudar Computação?

Frequentemente ouvimos ou proferimos a frase **Eu estou programado** em referência a tarefas que são executadas da mesma forma todos os dias - tarefas como arrumar a cama todos os dias, vir a Universidade todas as manhãs, seguir uma receita para cozinhar algum tipo de alimento, ou efetuar uma transação em um caixa eletrônico de banco.

Neste disciplina, você irá atuar o papel de projetista de receitas para outros seguirem. Contudo, haverão diferenças significativas entre programas que você projeta e receitas em um livro de cozinhar. Isto é, programas são projetados de tal forma que eles possam ser executados por um computador ao invés de uma pessoa. Desta forma, a linguagem que usamos para descrever programas é restrita pelos limites do vocabulário do computador. Apesar disso, veremos que muitas vezes que nossos programas serão capazes de modelar atividades que podem ser executadas por uma pessoa ao invés de um computador.

Neste sentido, programas de computador são *simulações* de uma solução de problemas por humanos que podem ser executados de forma mais eficiente ou mais precisa por um computador do que por uma pessoa. Por exemplo, considere o quase onipresente caixa eletrônico, o qual é programado para simular o trabalho de um caixa de banco na execução de tarefas simples, tais como saques e depósitos, para clientes. Desta forma, o programa é uma série de passos que são controlados através de um diálogo com o cliente.

Os caixas eletrônicos são utilizados por milhões de pessoas, todos os dias e em todos os bancos. Contudo, cada utilização deve ser executada de forma confiável e eficiente, a fim de que a integridade do processo bancário seja mantida. Portanto, o programa de computador que controla as atividades de um caixa eletrônico deve ser projetado com cuidado e precisão.

Estudar computação é mais do que aprender a programar a operação de uma máquina. É o estudo de um método do pensamento humano que tem evoluído por milhares de anos. A computação tem um enorme impacto em nossa cultura e em nossa vida diária. Nos últimos cinquenta anos, a computação proporcionou vários avanços nas ciências e profissões. Sem a computação, esse avanço não seria possível. Entender os fundamentos que caracterizam as aplicações de computadores para vida diária, organizando nosso processo de pensar para projetar soluções que possam ser resolvidas por um computador, e ser capaz de avaliar todo o seu potencial e limitações que essa atividade possibilita podem ser experiências agradáveis e estimulantes. É bom lembrar que a sociedade contemporânea não pode prescindir do computador e cada vez mais atividades do dia a dia da sociedade dependem do uso de sistemas de computação.

Este disciplina é projetada para introduzir a amplitude e vitalidade da área da computação. Você deve olhar para as quatro áreas principais na medida que abordamos as várias partes deste curso:

- A computação é firmemente baseada em lógica matemática. É também desejável o entendimento básico de funções matemáticas, visto que vários paradigmas computacionais utilizam a noção de função como modelo.
- O desenvolvimento de programas (software) confiável e efetivo envolve uma abordagem sistemática para a solução de problemas.
- Para projetar uma solução que utilize um computador necessitamos entender minimamente com um computador funciona. Um estudo mais detalhado do funcionamento de um computador será visto na disciplina de Introdução a sistemas digitais.
- O estudo de computação não pode ficar isolado do contexto social e profissional na qual a computação está inserida, e o qual proporciona sua razão de ser.

Esperamos que esta introdução tenha despertado seu interesse neste fascinante e dinâmico campo de estudo. Enquanto você for adquirindo um rigorosa experiência em programação, a mensagem de que computação é uma disciplina enorme, rica e dinâmica virá junto. Aprecie seu trabalho, e pense frequentemente a respeito da grande inter-relação que os vários exercícios nesta disciplina sugerem.

6.2 Computadores e Solução de Problemas

A variedade de problemas que podem ser solucionados por computador é muito grande. Muitos desses problemas, tais como o do exemplo do Caixa Eletrônico, são de natureza matemática e portanto envolvem a computação com números e a aplicação de princípios matemáticos. Outros são de natureza gráfica e envolvem a manipulação de objetos gráficos, tais como pontos, linhas, retângulos, e círculos. Outros problemas ainda envolvem a manipulação de texto em uma linguagem como Português e portanto usam caracteres do alfabeto e palavras como unidades básicas de informação. Contudo, muito frequentemente, os problemas mais interessantes e úteis requerem que o computador manipule informações em todos os três tipos desses domínios - matemática, gráficos e texto. Por exemplo, quando usando o computador para simular os movimentos de um jogo da velha, desejamos não somente ver a sequência de movimentos (representada como texto) mas também ver a impressão

do tabuleiro após cada jogada (representada graficamente). Uma razão fundamental para estudar problemas envolvendo ao mesmo tempo matemática, gráficos e processamento de texto é que os princípios e técnicas na solução dos problemas estudados em qualquer uma dessas áreas aplicam-se igualmente aos problemas das duas outras áreas. Isto sugere que existe uma base e um conjunto de ideias unificadas que orientam como computadores e programas se comportam. Estas ideias, por sua vez, influenciam nossa forma de pensar quando abordamos um problema específico para ser solucionado por um computador. Este conjunto unificado de ideias combina em um processo conhecido como solução algorítmica de problemas. Iniciaremos o estudo da solução algorítmica de problemas usando a teoria matemática de conjuntos e funções para auxiliar na explicação da estrutura básica dos computadores e programas de computador. Também veremos como é o projeto de um computador e como este projeto é usado para modelar as soluções para problemas algorítmicos. Utilizaremos uma abordagem no processo de abstração por meio da introdução de MAPS [Tucker et al.], uma metodologia de modelar soluções para problemas algorítmicos na forma de programas de computador. Desta forma, iniciamos o entendimento de como os três processos de teoria, abstração, e projeto se complementam entre si sendo indispensáveis no estudo de computação.

7 Problemas e Algoritmos

Os conceitos de problemas algorítmicos, algoritmos, linguagem algorítmica e comportamento de algoritmos são fundamentais para a atividade de solução de problemas com computadores e portanto são centrais para toda a computação. Desta forma necessitamos desenvolver um claro entendimento do que é um algoritmo e do que não é um algoritmo, e de como um problema algorítmico pode ser utilizado nas atividades de solução efetiva de problemas. As definições a seguir servem como um ponto de partida para nossa discussão:

Definição 2 *Um **problema algorítmico** é um problema qualquer que pode ser expresso por uma lista de instruções executáveis. Por executável queremos dizer uma instrução que um executor independente (homem ou máquina) pode executar em um modo passo a passo.*

Problemas algorítmicos possuem várias (muitas vezes um número infinito de) instâncias. Isto é, considere o problema de processar uma transação em um Caixa Eletrônico. Cada transação diferente representa uma instância particular para o problema do Caixa Eletrônico, visto que a transação pode ser um depósito ou um saque, e essa transação pode afetar o saldo em qualquer uma das milhares contas banco, e pode envolver uma quantia diferente de dinheiro.

Definição 3 *Um **algoritmo** é uma lista de instruções especificando uma definição precisa passo a passo de um processo que tem a garantia de terminar, após um número finito de passos, com a resposta correta para qualquer instância particular que possa ocorrer do problema algorítmico.*

Exemplos de problemas algorítmicos e algoritmos ocorrem em todos os lugares em nossa vida do dia a dia, ambos dentro e fora da computação. A Tabela 1 lista alguns exemplos adicionais.

Portanto, um algoritmo é sempre projetado com o objetivo de solucionar um problema em *todas* as suas instâncias. Contudo, para qualquer problema específico usualmente existem diversas alternativas de algoritmos que servem como soluções para esse problema. Por exemplo, uma receita específica para preparar costeletas fritas é uma das diversas alternativas

Tabela 1: Problemas Algoritmos e Algoritmos

Problema Algoritmo	**Algoritmo**
Preparando Costeletas Fritas	Qualquer receita para preparar costeletas de porco
Costurando um Vestido	Qualquer conjunto de instruções para fazer um vestido
Trocando um pneu de um carro	Qualquer conjunto de instruções para trocar um pneu
Vir a Universidade	Qualquer itinerário para chegar a Universidade

de algoritmos para solucionar o problema geral de preparar costeletas fritas. Um estudante de Ciência da Computação irá seguir um algoritmo bastante diferente para completar o seu curso do que um estudante do curso de Biologia ou Letras.

Apesar disso, cada um dos algoritmo da Tabela 1 exibe quatro características fundamentais dadas na definição:

- Exatidão - ("especifica uma descrição precisa")
- Efetividade - ("garantia de dar a resposta correta")
- Garantia de término - ("termina após um número finito de passos")
- Generalidade - ("para toda particular instância")

Alguns algoritmos, quando executados, podem ser completados em um curto período de tempo, enquanto outros podem tomar um tempo muito grande. Ainda assim todos algoritmos terminam. Portanto qualquer descrição passo a passo de um processo que não termina não é um algoritmo. Por exemplo, o processo de listar todos os números inteiros positivos, um por um, não é um algoritmo visto que ele não termina.

7.1 O Modelo de Computação Entrada-Processo-Saída

Como um ponto inicial, considere o modelo simplificado de computação mostrado na Figura 5. Neste modelo, uma computação é vista como tendo três partes: uma **entrada** (*input*), um **processo**, e uma **saída** (*output*). A entrada é uma coleção de informações que é requerida pelos passos do algoritmo para ser efetivamente executado; isto é, ela representa as informações que serão utilizadas durante a execução de um algoritmo. Por exemplo, a **entrada** para o caixa eletrônico pode ser qualquer número específico de conta corrente, tipo de transação, e quantidade de dinheiro. O **processo** é uma lista completa dos passos (instruções) que representa o algoritmo propriamente dito. A **saída** é o resultado que será obtido pela execução destes passos com os dados da entrada dada.

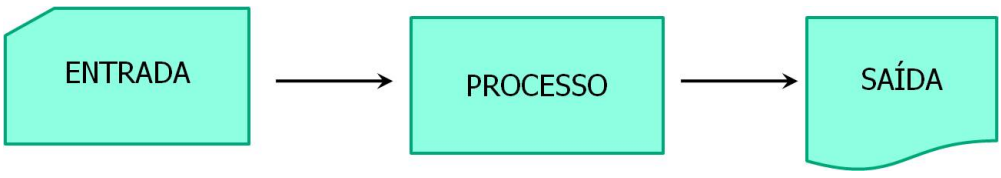

Figura 5: O Modelo de Computação Entrada-Processo-Saída.

Quando descrevemos os algoritmos (descrição do processo), na utilização do computador simplificado fazemos algumas suposições tácitas a respeito do comportamento do modelo entrada-processo-saída da Figura 5. Nossa primeira suposição é que a entrada deve ser finita. Nossa segunda suposição é que para computar qualquer coisa a respeito dessas entradas, o algoritmo necessita ler essas entradas - isto é, transferi-las fisicamente da parte da entrada para a parte do processo no modelo. Essa transferência física é chamada leitura, no linguajar da computação. Finalmente, para expressar os resultados da computação, devemos transferi-los da parte do processo no modelo para a parte da saída. Esta transferência física é chamada escrita.

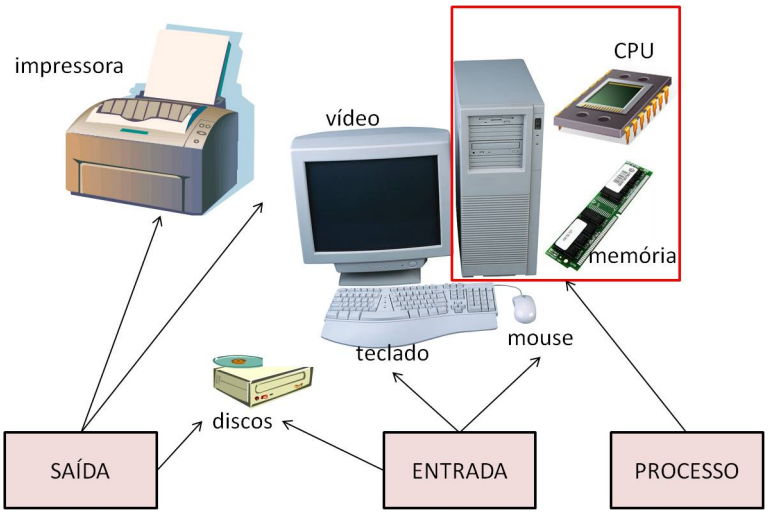

Figura 6: Um Sistema Computacional que implementa o Modelo de Computação Entrada-Processo-Saída.

Um Sistema Computacional que implementa o modelo entrada-processo saída é mostrado na Figura 6. Nesse sistema, utilizamos um computador para implementar um problema algorítmico, e a implementação usa o conceito de programa armazenado, isto é, as instruções descrevendo os passos individuais do processo e os dados necessários para sua execução são armazenados fisicamente na memória do computador (veja a figura 6). O modelo de computador utilizado anteriormente é uma versão simplificada desse sistema. A unidade central de processamento (CPU) contém os circuitos que executam os passos descritos no processo. Todos os dados de entrada e saída são manipulados pelas unidades de entrada e saída, como por exemplo, um teclado, um monitor de vídeo, unidades de disco, e uma impressora.

No nosso modelo, a **entrada**, **processo** e **saída**, a computação necessita de mais algumas premissas:

- Como a entrada de informações é fornecida para que o algoritmo possa funcionar;
- Como a saída de informações é executada;
- Tipos de instruções e operações que o **processo** pode executar.

Para ilustrar essas ideias, definiremos um computador simplificado. Ele é composto de um dispositivo de entrada, uma unidade central de processamento (CPU), composta de uma unidade lógico-aritmética (ULA) e uma unidade de controle (UC), um dispositivo de memória e um dispositivo de saída. As descrições dos dispositivos, operações e fluxos de dados visam apenas a descrever o funcionamento desse computador simplificado. Posteriormente, em Sistemas Digitais e Arquitetura de Computadores, esses dispositivos e operações serão descritos com detalhes.

Nosso computador tem a funcionalidade de ler um fluxo de informações de um dispositivo de entrada (p. ex. o teclado) e mover essas informações para a memória do computador. Pode também *imprimir* um fluxo de informações existentes na memória do computador para um dispositivo de saída (p. ex. a tela). A ULA faz operações ariméticas e comparações e a UC controla o fluxo de execução do algoritmo. A memória de um computador pode ser vista como uma grande tabela, onde cada posição dessa tabela tem um endereço específico. os endereços de memória do nosso computador simplificado variam de 10000 a 11000, sendo que o programa é armazenado a partir do endereço 10101 e o conteúdo das informações lidas ou geradas pelo algoritmo são armazenada a partir do endereço 10201.

Quando um fluxo de informação (p. ex. um número inteiro) é lido pela unidade de entrada ele será armazenado num dado endereço da memória do computador. Esse endereço da memória onde a informação será armazenada é dado quando alguma informação é lida pelo algoritmo ou quando alguma atribuição é feita. As informações armazenadas na memória podem ser acessados pela UC/ULA que pode fazer operações aritméticas ou lógicas com elas e quando for o caso armazenar o resultado num determinado endereço da memória. Quando queremos mostrar um resultado obtido pelo algoritmo, o computador acessa a posição da memória onde a informação está armazenada e a direciona para o dispositivo de saída.

Utilizamos os endereços das posições da memória para efetuar o armazenamento e acesso das informações. Para facilitar o armazenamento e o acesso das informações na memória, podemos fazer o uso de rótulos.

Na solução de um problema usando um computador, temos que descrever os passos da solução utilizando operações que a os dispositivos de entrada e saída, a UC/ULA e a memória consigam executar. Inicialmente precisamos de uma operação para ler (**input()**) as informações e outra para imprimir (**print()**) as informações. Antes de ler uma informação, necessitamos definir como ela será armazenada na memória. Para isso precisamos de um endereço na memória onde ela será armazenada e conhecer o tamanho da informação. Isso pode ser feito com a **utilização** de um rótulo para informação que será lida, associando um nome (rótulo) a um endereço da memória, e "reservando" uma quantidade de bytes para o armazenamento da informação na memória. O controle do fluxo de dados tanto interno como externo a CPU é feito pela Unidade de Controle. As instruções para a leitura e impressão de um número inteiro no nosso computador simplificado podem ser vistas a seguir:

1. `numero = int(input)()`
2. **print**`(numero)`

A primeira instrução lê (usando o dispositivo de entrada) um fluxo de dados, converte o fluxo lido num número inteiro e o armazena na posição da memória associada a `numero`. Como o número a ser lido é um número inteiro, o espaço reservado deve ser suficiente para armazenar um número inteiro. Na segunda instrução, o computador acessa a posição de memória associada a `numero`, lê o conteúdo e imprime o seu valor (usando o dispositivo de saída). Toda movimentação dos dados do dispositivo de entrada para a memória e da memória para o dispositivo de saída é coordenada pela UC.

Agora que temos uma ideia de como os dados podem ser movimentados do/para o computador, vamos analisar como fazer operações aritméticas e lógicas no computador. As operações aritméticas e lógicas são efetuadas pela ULA. Posteriormente, em **Sistemas Digitais**, a forma como essas operações são executadas serão estudadas com detalhes. Da mesma forma que na leitura e impressão de dados, omitiremos os detalhes e faremos uma interpretação mais simplificada de como operações aritméticas e lógicas são executadas no nosso computador.

Considere o problema de computar a soma de dois números inteiros. Inicialmente, esses números devem ser lidos e armazenados na memória do computador. Isso pode ser feito de forma semelhante ao descrito no algoritmo `LeImprime`. Uma vez lido os número, A UC *movimenta* esses números para a ULA, que efetua a soma e posteriormente, a UC *movimenta* esse resultado para uma dada posição (endereço) da memória.

1. `numero1 = int(input())`
2. `numero2 = int(input())`
3. `soma = numero1 + numero2`
4. `print(soma)`

As instruções de leitura, atribuição e impressão definem o espaço (endereço e tamanho) da memória onde as informações (números) e o resultado da soma dos números será armazenado. Usando um dispositivo de entrada, o algoritmo lê os valores dos números inteiros e os armazenam na memória. Para efetuar a soma, os conteúdos das posições da memória associadas a `numero1` e `numero2` são movimentadas para a ULA, a ULA computa a soma dos dois números, e a copia (atribui) o resultado na posição da memória associada a `soma`, e finalmente, a última instrução escreve o resultado do valor associado a `soma` num dispositivo de saída. Toda movimentação dos dados do dispositivo de entrada para a memória, da memória para a ULA, da ULA para a memória e da memória para o dispositivo de saída é coordenada pela UC.

A ULA também realiza operações lógicas. De maneira análoga, as instruções abaixo descrevem os resultados da operação lógica `and`. Se `clausula1` e `clausula2` forem `True`, será escrito na posição de memória associada a `resultado` o valor `True`. Para qualquer outra combinação de valores, `resultado` receberá o valor `False`.

1. `resultado = clausula1 and clausula2`
2. `print(resultado)`

A leitura e a movimentação de dados ocorre da mesma forma que no algoritmo `soma`.

Vamos detalhar um pouco mais as instruções de LeImprime, Soma e TabVerdade. Vamos transformar esses conjuntos de instruções em algoritmos. Vamos usar um formato para descrever os algoritmos usando a sintaxe do Python. A cada um dos passos vamos incluir um comentário `#` precedendo a instrução, descrevendo o passo que será executado pela instrução. Os comentários não são considerados na *execução* do algoritmo/programa. Posteriormente vamos descrever uma metodologia para o projeto e implementação de algoritmos/programas.

```
# Algoritmo: LeImprime
# Passo 1. Leia um número inteiro
numero = int(input())
# Passo 2. Imprima um número inteiro
print(numero)
# fim
```

Todas as instruções que iniciam com `#` são ignorada, pois são tratadas como um comentário para documentar o programa (veremos mais sobre comentários e documentação de programas no decorrer da disciplina). A instrução
`numero = int(input())`

lê um conjunto de caracteres do fluxo de entrada, converte para um número inteiro e atribui a `numero` (posição na memória associada a `numero`). A instrução
print`(numero)`
imprime o valor contido na memória associado a `numero` no dispositivo padrão de saída (geralmente a tela do computador).

```
# Algoritmo: Soma
# Passo 1. Leia dois números inteiros
numero1 = int(input())
numero2 = int(input())
# Passo 2. Calcule a soma dos dois números
soma = numero1 + numero2
# Passo 3. Imprima a soma dos números
print(soma)
# fim
```

No nosso curso, a descrição de um algoritmo para solucionar um problema usando um computador, começa com a palavra `Algoritmo`, seguido do nome do algoritmo. O conjunto de declarações do algoritmo são as informações necessárias para a execução do algoritmo e os passos são instruções que podem ser executados pela CPU (ULA + Unidade de Controle). A descrição do algoritmo termina com o comentário `# fim`. Na medida que novos conceitos e formas de representação das informações forem sendo introduzidas, ampliaremos a descrição dos nossos algoritmos/programas.

Vamos abordar agora a forma como as informações são representados num computador.

7.2 Conjuntos, Tipos de Dados e Variáveis

Analisando a definição de algoritmo e o exemplo do algoritmo da soma, um algoritmo pode ser visto como uma função matemática, e as informações do domínio dessa função podem ser vistos como elementos de um dado conjunto, para os quais estão associados um conjunto de operações.

No caso do algoritmo da soma, temos uma função que recebe dois números inteitos e computa e imprime a soma dos dois números.

$$f(a, b) = a + b, \forall a, b \in \mathbb{Z}$$

Vamos apresentar brevemente os conceitos de Teoria de Conjuntos que são necessários para o entendimento de como uma informação é representada num computador. Um estudo mais detalhado pode ser visto em Fundamentos de Teoria da Computação.

Coleções de elementos (números, objetos, etc.) estão em toda parte. Podemos pensar numa coleção de pessoas em uma sala de aula, a coleção dos dígitos e letras em uma placa de carro, ou a coleção dos nomes das empresas aéreas que utilizam o aeroporto de Guarulhos em São Paulo. Matematicamente, tais coleções são chamadas de *conjuntos*.

Definição 4 *Um* **conjunto** *é uma coleção bem definida de elementos. Geralmente os elementos compartilham algumas características comuns. Dizemos que um* **elemento** x **pertence** *a um conjunto S, quando o conjunto S contém o elemento x, e escrevemos que $x \in S$ para denotar que x pertence a S.*

Exemplo 1 - O conjunto $\{aa, ab, ba, bb\}$ é o conjunto de todos as possíveis strings de dos caracteres do alfabeto $\{a, b\}$.

Exemplo 2 - $M = \{abc, acb, bac, bca, cab, cba\}$ é o conjunto de todas as strings que são um particular arranjo, ou permutação, das três letras a, b e c. Note que existem seis tais permutações.

Exemplo 3 - $S = \{0, 1, 01, 001, 100, 0100\}$ é o conjunto de strings sobre o alfabeto $\{0, 1\}$. Existem muitos outros conjuntos de strings sobre este alfabeto, tais como $\{1, 10, 100, 1000, 10000\}$ e o próprio $\{0, 1\}$. Podemos colocar várias restrições em tais conjuntos. Por exemplo $\{0, 1, 01, 10, 11, 00\}$ é o conjunto de todas as strings sobre $\{0, 1\}$ que possuem comprimento 1 ou 2.

Exemplo 4 - $F = \{abcd, abdc, acbd, acdb, adbc, adcb, bacd, badc, bcad, bcda,$
$bdac, bdca, cabd, cadb, cbad, cbda, cdab, cdba, dabc, dacb, dbac, dbca, dcab, dcba\}$ é o conjunto de todas as permutações das quatro letras a, b, c, e d. Isto também pode ser escrito como $F = \{p | p$ é uma permutação de $abcd\}$. Note que existem vinte e quatro permutações.

Alguns conjunto são de tamanho finito, enquanto outros tem um número infinito de elementos.

Definição 5 *A* **cardinalidade** *de um conjunto finito é o número de elementos do conjunto. O conjunto vazio não possui nenhum elemento, então ele tem cardinalidade* 0.

Existem quatro conjuntos que possuem uma importância especial tanto em computação como em matemática:

$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$	Números Naturais
$\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$	Inteiros
$\mathbb{R} = \{x \mid -\infty < x < \infty\}$	Números Reais
$\mathbb{Z}_n = \{0, 1, 2, 3, \ldots, n-1\}$	Inteiros módulo n

Os três primeiros são infinitos. O último é finito e tem cardinalidade n, pois o conjunto $\mathbb{Z}_2 = \{0, 1\}$ tem cardinalidade 2; o conjunto $\mathbb{Z}_{10} = \{0, 1, \ldots, 9\}$ tem cardinalidade 10, e assim sucessivamente.

Pelo fato de que o tamanho da memória de um computador é finito, o armazenamento de informações num computador também será finito. No caso das informações de um algoritmo serem números inteiros, temos que delimitar uma faixa (finita) na qual os números poderão variar. Essa faixa depende do número de bytes que serão utilizados para representar um número inteiro. Se o algoritmo utilizar números reais, a situação é ainda mais complexa, pois além da parte inteira, teremos que definir quantas casas serão utilizadas para representação do número.

Como em matemática, cada conjunto possui um conjunto definido de operações. Em computação, usamos termos específicos para para definir um conjunto de elementos com o conjunto de operações associado.

Definição 6 *Um* **tipo de dado** *é um conjunto de todos os possíveis valores junto com um conjunto de operações que podem ser executados com os elementos do conjunto. Algumas vezes, um tipo de dado é referido como* ***classe****.*

Não é requerido que cada operação seja válida para todos elementos do conjunto. Um exemplo é quando o tipo de dado inclui a divisão, a qual não é valida quando o divisor é zero.

Além dos naturais $\mathbb{N}$, inteiros $\mathbb{Z}$ e reais $\mathbb{R}$, outros tipos serão de interesse imediato para nós: C o conjunto dos caracteres (identificamos C com o conjunto de caracteres ASCII discutido anteriormente), B o conjunto de booleanos (o conjunto consiste de somente dois valores **true** e **false**, S o conjunto das *strings* e o tipos abstratos que podem ser construídos a partir dos tipos iniciais.

Tipo de Dados	Símbolo	Python	Operações
Inteiros	$\mathbb{Z}$	`int`	`+, -, *, //,%`
Reais	$\mathbb{R}$	`float`	`+, -, *, /`
Caracteres	Σ	`char`	
Strings	S	`str`	`+, etc`
Booleanos	$\mathbb{B}$	`bool`	`and, or, not, in`

Tabela 2: Conjuntos e Tipos de Dados

Como notamos anteriormente, a cardinalidade dos conjuntos matemáticos $\mathbb{N}$, $\mathbb{Z}$ e $\mathbb{R}$ é infinita. Isto é, podemos passar a vida toda contando e nunca acabarão os naturais, inteiros e reais. No caso dos $\mathbb{R}$, é mais complexo ainda, pois se trata de um conjunto **não enumerável**. Logo a representação desses conjuntos num computador ficarão restritos a conjuntos finitos, cuja cardinalidade dependerá da quantidade de bytes para representar cada um dos elementos desses conjuntos. Essa cardinalidade depende da arquitetura do computador e da linguagem que estiver sendo utilizada. No caso do conjunto dos números reais, tem mais uma complicação, pois não é possível representar um número potencialmente infinito de dígitos (tal como $1/3 = 0.33333...$) em um computador finito. A discussão em detalhes desse assunto será abordada em **Introdução a Sistemas Digitais**.

Além dos números decimais representados por uma dízima, também necessitamos esclarecer que os números irracionais como π, e, $\sqrt{2}$, etc, não possuem uma representação decimal exata, ou seja, não podem ser representados de forma exata por um número p/q. Com isso, nos computadores, os números $\mathbb{R}$ são representados com um faixa finita dos número racionais $\mathbb{Q}$. E na representação dos números racionais, temos que definir o número de casas decimais (precisão) que será utilizado na representação.

Vários problemas computacionais envolvem a manipulação e o processamento de conjuntos de caracteres. Em computação, denominamos **string** como sendo uma sequência qualquer de caracteres contíguos de um conjunto universal predefinido, conhecido como alfabeto. O alfabeto padrão em computação é conhecido como o conjunto de caracteres ASCII (American Standard Code for Information Interchange). Outros alfabetos existem, incluindo o EBCDIC para mainframes IBM. Como veremos, o tipo **string** será muito utilizado na solução de problemas computacionais.

A Tabela 2 ilustra uma tabela com o resumo dos tipos de dados básicos utilizados na linguagem `Python` com algumas das operações definidas para estes tipos.

Na Tabela 2, as operações associadas aos tipos de dados são definidas como: `*` representa o produto, `/` representa o quociente, e `%` o resto da divisão inteira; `+` concatena duas strings, `=` faz uma cópia de uma `string`; e `and`, `or`, e `not` são conectivos lógicos que serão apresentados nas próximas aulas. Além disso, os operadores relacionais (`>`, `>=`, `=`, `<=`, `<`, `==`) também podem ser aplicados para cada um desses tipos.

Quando iniciarmos a codificação programas em `Python`, a representação dos elementos dos conjuntos matemáticos serão mapeados aos elementos de tipos como `int` e `float`. Contudo, estes tipos não são equivalentes aos conceitos matemáticos dos números inteiros e reais, como descrito acima, precisamente pelo fato de que os últimos são conjuntos infinitos. Desta forma, em `Python`, a variação do tipos `int` é restrito, como também a precisão do tipos `float`. Iremos discutir essas restrições mais tarde.

Como vimos, o conjunto de valores que os elementos de um conjunto podem assumir definem o seu tipo. Da mesma forma que em matemática exploramos a estrutura de um conjunto associado com um conjunto de operações que podem ser executados com o elementos do conjunto, temos que diferentes tipos podem ser manipulados de formas distintas.

Se estivermos trabalhando com inteiros, não faz sentido falarmos da parte fracionária numa divisão. Desta forma cada tipo terá um conjunto correspondente de manipulações válidas ou operações as quais podem ser executadas nos elementos daquele tipo.

O conceito de variável utilizado em matemática será usado para representar informações armazenadas na memória do computador. Com isso, por meio de um símbolo podemos representar com uma noção precisa um determinado elemento de um conjunto previamente definido.

Definição 7 *Uma* **variável** *é um símbolo que está associado com um único valor, o qual por sua vez pode ser qualquer valor de um conjunto bem definido. Durante os passos de uma computação, o valor da variável pode mudar de um elemento para outro elemento do conjunto. O conjunto para o qual uma variável particular está definida é chamado de* **tipo**. *Em alguns contextos de programação, a ideia de uma variável e seu tipo é referida como um* **objeto** *e sua* **classe**.

Se olharmos na descrição do processo do algoritmo do `Soma`, podemos ver várias referências para a computação de um valor (p.e., `soma = numero1 + numero2;`). A atribuição implícita dessas instruções usa o conceito de variável, que é uma entidade de dar nomes que podem receber um valor (`numero1`, `numero2`, e `soma` são todas variáveis em nosso exemplo). Embora nesta computação cada variável recebe um único valor, em várias situações o valor de uma variável pode mudar diversas vezes durante a computação. Além disso, o valor de uma variável será diferente e, em geral, imprevisível de uma instância de execução do algoritmo para a próxima. Por exemplo, caso executemos o algoritmo `Soma` para a entrada `8 9` teremos as variáveis `numero1 = 8`, `numero2 = 9` e `soma = 17`. Para resolver problemas usando computadores, é fundamental que entendamos variáveis e os *conjuntos* relacionados de valores que elas podem ter.

Os nomes atribuído a variáveis devem respeitar a sintaxe exigida pela linguagem de programação. No caso dos algoritmos, nossa variáveis sempre começam com uma letra e podem usar determinados símbolos especiais na sua formação. Sempre é recomendável utilizar nomes que sejam significativos ao invés de usar um único caractere para representar uma variável.

Para um problema para computar o perímetro e a área de um dado quadrado, podemos utilizar a variável `lado` para representar o lado do quadrado e as variáveis `perimetro` e `area` para representar o perímetro e a área do quadrado. Neste nosso caso, todas as variáveis são do tipo `float`.

Na linguagem `Python` não é necessário fazer a declaração de variáveis, pois ela usa tipos dinâmicos, e nesse caso, a *atribuição* ocorre quando a variável é referenciada, por exemplo. quando for atribuída uma valor a variável. Com isso, no referenciamento da variável é associado o tipo de dados a que ela pertence o que possibilita a "reserva" de espaço na memória do computador para armazenar as informações referentes a essa variável.
Exemplos de nomes de variáveis em `Python`.

```
bool          fato
long          divida_publica # sublinhado
float         taxaPagamento  # Maiúscula
complex       voltagem
char          code, tipo     # estilo ruim
int           a, b           # estilo ruim
```

7.3 Exemplos

Nos exemplos abaixo ilustraremos como o modelo de computação Entrada-Processo-Saída num computador simplificado funciona. A discussão de como projetar e implementar um algoritmo será visto mais adiante.

Os instruções constantes na descrição dos algoritmos são executados na mesma ordem em que foram escritas. Só podemos executar uma nova instrução após as anteriores terem sido completadas. Além disso, temos que cada instrução individual é efetiva, ou seja, ela só inclui ações que que um computador possa executar. Por exemplo, assumimos que print está dentro do repertório das capacidades do computador. Desta forma, sempre que uma instrução contendo a palavra print for executada, a informação de saída aparecerá no monitor (ou outro dispositivo de saída), independentemente de onde esta instrução ocorra dentro do algoritmo.

Essas funcionalidades são fundamentais para todos os algoritmos que são desenvolvidos para computadores. Computadores executam as instruções de uma maneira obediente, começando sempre com a primeira, e só irão executar instruções que sejam inteligíveis para eles - isto é, instruções as quais descrevam tarefas que estejam dentro dos seus repertórios.

1. Considere o problema de somar dois números inteiros. Neste caso, a **entrada** são dois números inteiros quaisquer. A **saída** é o resultado da soma dos dois números, e o **processo** é a operação da soma de dois número inteiros.

```
1  # Algoritmo: Soma
2  # descrição das variáveis utilizadas
3  # int numero1, numero2, soma
4  # Passo 1. Leia dois números inteiros
5  numero1 = int(input())
6  numero2 = int(input())
7  # Passo 2. Calcule a soma dos dois números
8  soma = numero1 + numero2
9  # Passo 3. Imprima a soma dos números
10 print(soma)
11 # fim
```

   Inicialmente o computador armazena o processo (lista de instruções do algoritmo) na memória do computador a partir do endereço 10101 e a unidade de controle é a responsável pelo gerenciamento da execução de cada linha do algoritmo. Os valores das informações serão armazenadas a partir do endereço de memória 10201, obedecendo a ordem em que os rótulos forem declarados.

   Como vimos na seção7.1, para uma melhor compreensão do algoritmo, usamos comentários para descrever o funcionamento dos algoritmos. Com isso, todas as linhas que são precedidas pelo caractere “#”são ignoradas e não serão processadas pelo nosso computador.

   A primeira instrução do algoritmo pela primeira linha sem comentário, na linha 10101, e ele termina com a última linha sem comentário, na linha 10105. A execução começa com a primeira instrução do algoritmo, a linha 10101. A UC gerencia a execução de cada uma das linhas, e a mudança para a linha seguinte, caso ela exista, indica que a instrução foi completada. Quando não existir mais nenhuma linha sem comentários, o algoritmo termina.

Ao executar as linha 10101 e 10102, o algoritmo "reserva" duas posições na memória para armazenar os conteúdos de `numero1` e `numero2` e usando o dispositivo de entrada lê dois inteiros e armazena nas variáveis `numero1` e `numero2`. No nosso caso, a posição 10201 será utilizada para armazenar `numero1`, a posição 10202 para armazenar `numero2`.

10001	....
10002	...
...	...
10101	`numero1 = int(input())`
10102	`numero2 = int(input())`
10103	`soma = numero1 + numero2`
10104	`print(soma)`
10105	
...	...
10201	
10202	
10203	
...	...

A **entrada** pode ser dada por dois números inteiros quaisquer. Ao executar a linha 10102, o dispositivo de entrada está habilitado para ler dois números inteiros e armazená-los na memória nas posições 10201 e 10202. Se fornecermos dois números inteiros tais como

```
4, 6
```

os números serão armazenados nas posições da memória 10201 e 10202, e estarão associados aos rótulos `numero1` e `numero2`, respectivamente.

10001	....
10002	...
...	...
10101	`numero1 = int(input())`
10102	`numero2 = int(input())`
10103	`soma = numero1 + numero2`
10104	`print(soma)`
10105	
...	...
10201	`4`
10202	`6`
10203	
...	...

Ao executar a linha 10103, o conteúdo das posições da memória 10201 (`numero1`) e 10202 (`numero2`) são copiados na ULA que executa a operação "`+`"e o resultado é armazenado na posição da memória 10203 (`soma`).

A execução da instrução da linha 10103 é mais complexa, mas para entendermos como o modelo de computação e um computador simplificado, a forma apresentada

é suficiente. Como dissemos, posteriormente esse assunto será estudado na disciplina de Sistemas Digitais.

A execução da instrução da linha 10104 acessa o conteúdo da posição da memória 10203 (`soma`) e imprime esse valor no dispositivo de saída.

10001	....
10002	...
...	...
10101	`numero1 = int(input())`
10102	`numero2 = int(input())`
10103	`soma = numero1 + numero2`
10104	`print(soma)`
10105	
...	...
10201	4
10202	6
10203	10
...	...

Como a última instrução está na linha 10104, o algoritmo termina.

Se as instruções das linha 10101 a 10104 (**processo**) forem executadas corretamente a **saída** para a entrada dada acima será

```
10
```

Juntos, essa entrada e essa saída representam uma **instância** particular do problema da soma.

O nosso algoritmo `Soma` recebe dois números inteiros, $(numero1, numero2) \in \mathbb{Z} \times \mathbb{Z}$ e retorna um um número inteiro $soma = numero1 + numero2$. Temos que o projeto e o desenvolvimento de algoritmos está diretamente relacionado com a matemática, pois o algoritmo `Soma` pode ser interpretado como sendo uma função

$$Soma(numero1, numero2) = numero1 + numero2$$

$$f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$$

2. Dado um quadrado de um determinado lado, quais seriam os passos necessários para computar o perímetro e a área desse quadrado? A **entrada** é um dada por um número real representando o lado do quadrado e a **saída** é o perímetro é dado pela soma dos quatro lados, e a área do quadrado é dada pela quadrado do lado.

   O algoritmo `Perímetro_Área` descreve uma solução para esse problema no modelo de computação.

```
1 # Algoritmo Perímetro_Área
2 # descrição das variáveis utilizadas
3 float lado, perimetro, area
4 # Passo 1. Leia o lado do quadrado
```

```
5  lado = float(input())
6  # Passo 2. Compute o perímetro e a área do quadrado
7  # Passo 2.1. Compute o perímetro do quadrado (4 * lado)
8  perimetro = 4 * lado
9  # Passo 2.2. Compute a área do quadrado (lado * lado)
10 area = lado * lado
11 # Passo 3. Imprima o Perímetro e a Área do quadrado
12 print(perimetro, area)
13 # fim
```

Ao "carregarmos"o nosso algoritmo para o computador simplificado, todos os comentários são descartados e somente as instruções executáveis são copiadas na memória. A primeira e a última linha do algoritmo somente sinalizam o início e o final das instruções.

10001	....
10002	...
...	...
10101	lado = `float(input())`
10102	`perimetro = 4 * lado`
10103	`area = lado * lado`
10104	`print(perimetro, area)`
10105	
...	...
10201	
10202	
10203	
...	...

A **entrada** pode ser um número real qualquer (vamos usar a forma de número real do Python), tal como

```
3.5
```

Ao executar a instrução da linha 10101, é feita uma "reserva"de um espaço na memória (10201) para armazenar o conteúdos de `lado`. Com isso o valor `3.5` será armazenado na posição 10201 da memória (endereço associado a `lado`). A execução das linhas10102 e 10103 "reserva"dois espaços na memória (10202, 10203) para armazenar os conteúdos de `perimetro` e `area`. Além disso, essas instruções buscam o valor associado a variável `lado` armazenado na memória na posição 10201 e usando a unidade lógica e aritmética computa os valores de `perimetro` e `area` e armazenam os valores `14.00` e `12.25` nas posições 10202 e 10203, respectivamente.

Se as instruções das linhas 10101 a 10104 (**processo**) forem executadas corretamente a **saída** (linha 10104) para a entrada dada acima será

```
14.00 12.25
```

Juntos, essa entrada e essa saída representam uma **instância** particular do problema do perímetro e área de um quadrado.

10001	....
10002	...
...	...
10101	lado = `float(input())`
10102	`perimetro = 4 * lado`
10103	`area = lado * lado`
10104	`print(perimetro, area)`
10105	
...	...
10201	3.5
10202	14.00
10203	12.25
...	...

3. Considere o problema de computar a média entre três notas variando de 0.0 a 10.0. Neste caso, a **entrada** são três números reais maiores ou iguais a zero e menores ou iguais a dez. A **saída** é a média das três notas da entrada, um número real, e o **processo** é a descrição dos passos que são necessários para executar o algoritmo.

```
1  # Algoritmo Média
2  # descrição das variáveis utilizadas
3  # float nota1, nota2, nota3, soma, media
4  # Passo 1. Leia as notas nota1, nota2, nota3
5  nota1 = float(input())
6  nota2 = float(input())
7  nota3 = float(input())
8  # Passo 2. Compute a soma e a média das notas
9  # Passo 2.1. Compute a soma das notas (nota1 + nota2 + nota3)
10 soma = nota1 + nota2 + nota3
11 # Passo 2.2. Compute a média das notas, soma das notas dividido por 3.0
12 media = soma/3.0
13 # Passo 3. Imprima a média das notas
14 print(media)
15 # fim
```

A **entrada** pode ser dada por três números reais entre 0.0 e 10.0 quaisquer, tais como

```
5.0 6.0 5.6
```

No nosso computador simplificado, utilizamos um mesmo espaço para armazenar um número inteiro, um número real ou um caractere. Num computador real o tamanho dessas informações é diferente. Além disso, é bom lembrar que quando estamos trabalhando com inteiros, `10//3 = 3`, mas se estivermos trabalhando com números reais `10.0/3.0 = 3.333...`. Além disso, como os números reais contém os números inteiros ($\mathbb{R} \supset \mathbb{Z}$), o resultado da divisão de um inteiro com um número real será um número real (`10/3.0 = 3.333...`).

Como no exemplo anterior, o "carregamento" do algoritmo `Média` na memória do nosso computador descarta todos os comentários (linhas que iniciam com `#`).

10001	....
10002	...
...	...
10101	`nota1 = float(input())`
10102	`nota2 = float(input())`
10103	`nota3 = float(input())`
10104	`soma = nota1 + nota2 + nota3`
10105	`media = soma/3.0`
10106	`print(media)`
10107	
...	...
10201	5.0
10202	6.0
10203	5.6
10204	16.6
10205	55.333...
...	...

Se as linhas 10101 a 10106 (**processo**) forem executadas corretamente a **saída** (linha 10107) para a entrada dada acima será

```
55.333...
```

Juntos, essa entrada e essa saída representam uma **instância** particular do problema da soma.

4. Considere o problema de ler uma dada palavra e imprimir o primeiro caractere da palavra. Como usaremos um computador para implementar o algoritmo, temos que especificar melhor o conceito de palavra. Os computadores utilizam alfabetos específicos, com caracteres que podem ser visíveis ou não. Para simplificar, consideraremos uma **palavra** como sendo um conjunto de caracteres $c_1c_2 \ldots c_k$, onde cada c_i é uma letra do alfabeto da língua portuguesa. Toda palavra é precedida de um espaço em braco (caractere branco) e após a palavra também tem um espaço em branco. De posse da definição de uma palavra temos que a **entrada** será uma palavra, que é dada por um conjunto de quatro (4) caracteres. A **saída** do nosso problema será a impressão do primeiro caractere da palavra lida.

```
1 # Algoritmo Caractere
2 # descrição das variáveis utilizadas
3 # char palavra, caractere
4 # Passo 1. Leia os caracteres de uma palavra
5 palavra = input()
6 # Passo 2. Compute o primeiro caractere da palavra
7 caractere = palavra[0]
8 # Passo 3. Imprima o primeiro caractere da palavra
```

```
9  print(caractere)
10 # fim
```

A **entrada** é uma palavra qualquer (de acordo com a definição acima), dada por um conjunto de 4 caracteres, tal como

```
roma
```

10001	....
10002	...
...	...
10101	`palavra = input()`
10102	`caractere = palavra[0]`
10103	`print(caractere)`
10104	
...	...
10201	r
10202	o
10203	m
10204	a
10205	
...	...

Se as linhas 10101 a 10103 (**processo**) forem executados corretamente (lembrando que os comentários serão descartados na execução do algoritmos) a saída para a entrada dada acima será

```
r
```

Juntos, essa entrada e essa saída representam uma **instância** particular do problema de computar o tamanho de uma palavra.