**Apresentação - Lógica de Programação**

A disciplina de Lógica de Programação representa a entrada do aluno na área de desenvolvimento de sistemas e correlatos. Seu conteúdo abrange os conceitos e ferramentas básicas de programação, como variáveis e estruturas de controle, que capacitam o aluno a solucionar problemas computacionais e construir programas utilizando essas ferramentas básicas existentes em qualquer linguagem. É uma disciplina de vital importância, pois seu conteúdo serve de alicerce para uma série de outras disciplinas que abordam aspectos mais avançados de programação e que serão vistas na sequência do curso.

Acompanhe a apresentação da disciplina, elaborada pelo Professor Gilberto Aves Pereira:

[Apresentação Lógica de Programação](https://www.youtubeeducation.com/watch?v=L5l2bjfRi4g).

**Objetivos gerais**

Desenvolvimento do Raciocínio Lógico;

Aprender a descrever de forma gráfica uma solução lógica;

Entender os Conceitos Envolvidos em Lógica de Programação;

Ser capacitado a utilizar as ferramentas básicas de lógica de programação abaixo:

* Variáveis e Comandos de Entrada e Saída;
* Estruturas de Controle de Fluxo (Sequencial, Seleção e Repetição);
* Funções;
* Estruturas de Armazenamento - Vetores.

**Parte 1 -** [**Introdução - Algoritmos e Fluxogramas**](https://sites.google.com/faculdadeimpacta.com.br/lo-p1?pli=1&authuser=2) **-** [**Video**](https://www.youtubeeducation.com/watch?v=KWTQJxJ_ZPU)

**Resumo**

Nesse texto discutimos os conceitos básicos envolvidos quando falamos inicialmente sobre lógica de programação: Algoritmos, o funcionamento de um software, Linguagem de Programação e por fim falamos um pouco sobre a ferramenta que usaremos em nosso curso - o Fluxograma.

**Algoritmo**

O termo algoritmo pode ser visto desde o século IX. Foi nesta época que o cientista, astrônomo e matemático persa Abū ‘Abd Allāh Muhammad ibn Mūsā al-Khwārizmī usou pela primeira vez o termo para indicar regras de operações aritméticas utilizando algarismos indo-arábicos (DIERBACH 2012).

No século XII, Adelardo de Bath traduziu o termo para o latim Algorithmi. De lá para cá, o termo evoluiu bastante, incluindo todos os procedimentos definidos para resolver problemas ou realizar tarefas.

DIERBACH (2012) A formalização da noção de algoritmo ocorreu em 1936 com os trabalhos de Alan Turing e Alonzo Church, que desenvolveram independentemente os modelos de Máquinas de Turing e Cálculo Lambda.

Do ponto de vista computacional, um algoritmo pode ser visto como um conjunto de regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de passos.

DIERBACH (2012) Donald Knuth, um dos pesquisadores mais respeitados em algoritmos, indica uma lista

de cinco propriedades que são requisitos para algoritmos:

**Finitude**: um algoritmo deve sempre terminar após um número finito de etapas (ou passos).

**Definição**: cada passo de um algoritmo deve ser definido com precisão. As ações a serem executadas deverão ser especificadas rigorosamente e sem ambiguidades.

**Entrada**: valores que são dados ao algoritmo antes que ele inicie.

**Saída**: os valores resultantes das ações do algoritmo a partir de uma determinada entrada.

**Eficácia**: todas as operações a serem realizadas pelo algoritmo devem ser suficientemente básicas para poderem, em princípio, ser feitas com precisão e em um período de tempo finito por um homem usando papel e lápis.



A formalidade pode ser conseguida com o uso de lógica. Assim, vamos exigir que um algoritmo seja uma sequência lógica de passos com começo, meio e fim.

Comumente, esta lógica é conhecida como lógica de programação e isto ocupará grande parte de nossa disciplina.

**Funcionamento de um Software**

De uma maneira simplificada, um software para funcionar precisa de um ambiente chamado Sistema Digital. O sistema digital é composto minimamente por uma unidade de processamento de instruções (UCP), memória (volátil - RAM), e de dispositivos de entrada e saída. As informações dentro do sistema digital estão em formato binário (zeros e uns).

O software é um conjunto de instruções para o processador que fica armazenado na memória. Esse conjunto de instruções basicamente fazem o seguinte:

1 - Obtém dados de algum dispositivo de entrada (teclado, mouse, HD, placa de rede,etc) e armazena na memória;

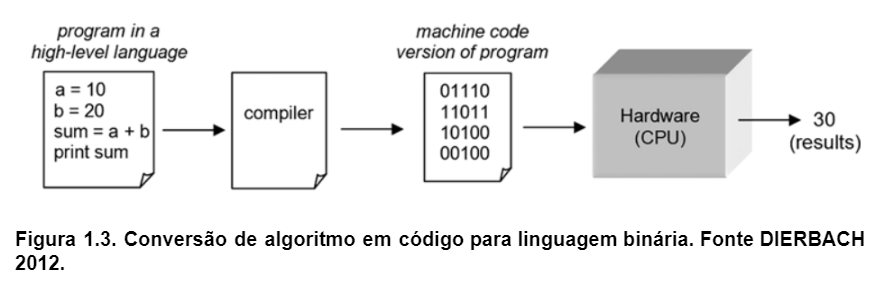
2 - Com as informações na memória efetua cálculos - o que chamamos de processamento, armazenando esses resultados na memória;

3 - Por fim, envia esses resultados desses cálculos da memória para algum dispositivo de saída (impressora, tela, placa de rede,etc).



**Sobre Linguagens de Programação e Algoritmos**

A Linguagem de Programação auxilia no processo de conversão para a linguagem binária do processador, através de uma série de regras que eliminam as ambiguidades.



Algoritmos resolvem qualquer problema. As Linguagens resolvem problemas utilizando os recursos do Sistema Digital (dispositivos de entrada, memória, processamento e dispositivos de saída). Nem todos Algoritmos podem ser implementados por Linguagens.

**O processo para resolver problemas computacionais**

A resolução de um problema computacional não envolve simplesmente o ato de programar um computador. Ele é um processo, sendo que a programação é apenas um dos passos. Antes de escrever um programa é preciso entender e pensar a solução do problema, depois desenvolver o projeto, escrever e testar a solução encontrada.

Uma vez que o algoritmo tenha sido definido, podemos simulá-lo com ferramentas como Scratch, VisuAlg ou Raptor, representá-los em Fluxograma ou implementá-lo diretamente em alguma linguagem de programação (Python, C, C++, Java, Pascal, PHP, dentre outras). Nossa disciplina será focada em Fluxograma.

**Fluxograma**

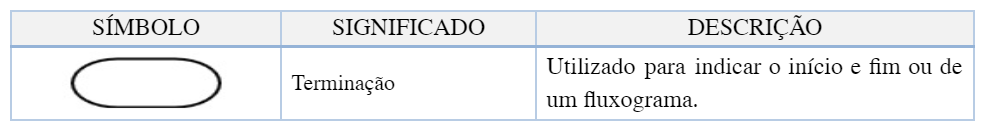
Fluxograma é uma forma padronizada e eficaz para se representar os passos lógicos de um determinado processo (algoritmo), é uma técnica de representação gráfica na qual se utilizam símbolos previamente convencionados, permitindo a descrição clara e precisa do fluxo, ou sequência de um processo, bem como sua análise e redesenho.

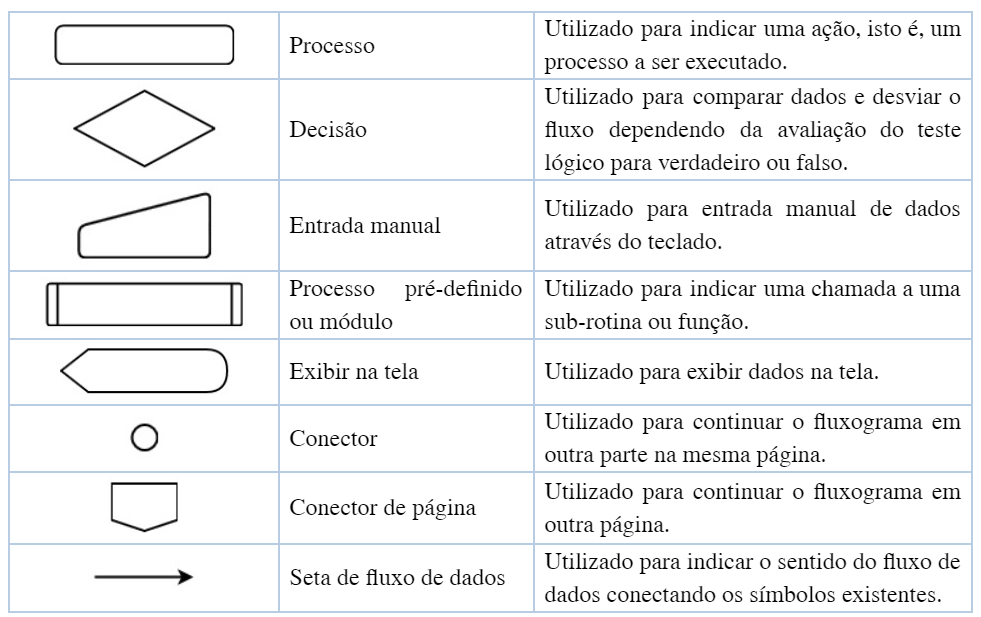
Vantagens: maior clareza na visualização do fluxo de execução (“imagens valem mais do que mil palavras”).

Desvantagens: requer conhecimento de convenções gráficas e dificuldade para fazer correções.

O fluxograma é representado por símbolos e cada um é utilizado para uma finalidade específica, o que torna o fluxograma mais fácil de compreender, pois a representação segue um padrão pré-estabelecido.

Neste curso, usaremos apenas uma pequena parcela dos símbolos disponíveis para uso nos fluxogramas, mostrados na tabela a seguir:

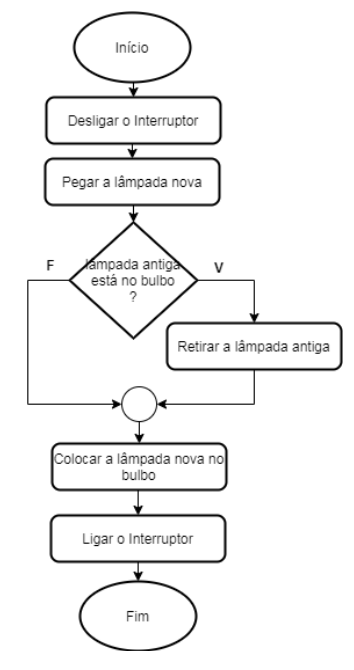




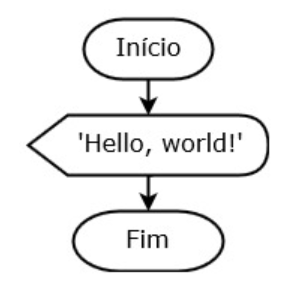
Fluxograma para troca de uma lâmpada. Nesse exemplo, estamos usando um símbolo diferente e novo em relação ao exemplo anterior, o símbolo de decisão.



Este símbolo nos permite criar dois caminhos em nosso fluxograma. Nesse caso, um dos caminhos não fazemos nada e o outro retiramos a lâmpada do bulbo. Tomamos o caminho dependendo da resposta a pergunta - Verdadeiro ou Falso.



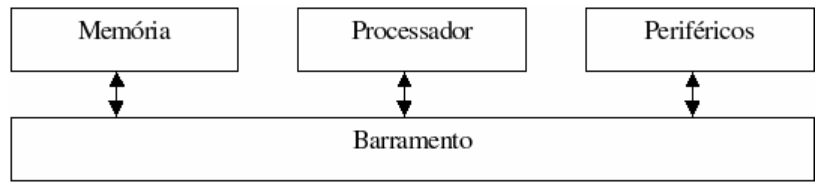
Para exibir uma mensagem na tela, utilizamos o símbolo Exibir. Esse símbolo é uma Saída de Dados, que pode ser utilizado para mostrar o resultado de um processo, um nome ou até mesmo para pedir para que o usuário digite alguma coisa.



Ferramenta para realização dos fluxogramas. - <https://www.drawio.com/>

**Introdução à Arquitetura de Computadores**

**Componentes básicos de um computador**



**Processador (ou microprocessador)**

É um circuito integrado (ou chip). É considerado o cérebro do computador. É ele que executa os programas, faz os cálculos e toma as decisões, de acordo com as instruções armazenadas na memória.

Os microprocessadores formam uma parte importantíssima do computador chamada de UCP (Unidade Central de Processamento), ou em inglês CPU (Central Processing Unit). Antes da existência dos microprocessadores, as CPUs dos computadores eram formadas por um grande número de chips, distribuídos ao longo de uma ou diversas placas. Um microprocessador nada mais é que uma CPU inteira, dentro de um único chip.

Ligando-se um microprocessador a alguns chips de memória e alguns outros chips auxiliares, construiu-se um computador inteiro em uma única placa de circuito, chamada placa mãe dos microcomputadores.

A CPU realiza as seguintes tarefas:

1 - Busca e executa as instruções existentes na memória. Os programas e os dados que ficam gravados no disco (rígido ou disquete) são transferidos para a memória. Uma vez estando na memória a CPU pode executar os programas e processar os dados.

2 - Comanda todos os outros chips do computador

A CPU é composta por:

**Unidade lógica e aritmética (ULA) -**  Assume todas as tarefas relacionadas às operações lógicas (and, or, not, etc.) e aritméticas (adições, subtrações, ...)

**Unidade de controle (UC) -** Assume toda a tarefa de controle das ações a serem realizadas pelo computador, comandando todos os demais componentes de sua arquitetura. É a UC que deve garantir a correta execução dos programas e a utilização dos dados corretos nas operações que as manipulam.

**Os registradores -** A CPU contém internamente uma memória de alta velocidade que permite o armazenamento de valores intermediários ou informação de comando. Esta memória é composta por registradores (ou registros) na qual cada registro possui uma função própria. Um registro memoriza um número limitado de bits, geralmente uma palavra de memória. Os registros mais importantes:

* contador de programa (PC) que aponta para a próxima instrução a executar;
* registro de instrução (IR) que armazena a instrução em execução;
* outros registros que permitem o armazenamento de resultados intermediários.

**Clock**

Clock é um circuito oscilador que tem a função de sincronizar e ditar a medida de velocidade de transferência de dados entre duas partes essenciais de um processamento, por exemplo, entre o processador e a memória principal. Esta frequência é medida em ciclos por segundo, ou hertz .

**Memória**

Constitui de um conjunto de circuitos capazes de armazenar os dados e os programas a serem executados pela máquina. Temos as seguintes categorias de memória:

a ) **A memória principal (ou memória de trabalho)**

É onde normalmente devem estar armazenados os programas e dados a serem manipulados pelo processador. Este tipo de memória aparece como um conjunto de chips que são inseridos na placa mãe do computador.

b) **A memória secundária (ou memória de massa)**

Não é acessada diretamente pela CPU. O acesso é feito através de interfaces ou controladoras especiais. É uma memória do tipo permanente. Possui alta capacidade de armazenamento e um custo menor que o da memória principal. A memória secundária não é formada por chips, e sim por dispositivos que utilizam outras tecnologias de armazenamento (magnética ou óptica). Exemplos: disco rígido, disquete, fita magnética e cd-rom.

**Tipos de memória**

Os chips de memória podem ser divididos em duas grandes categorias:

**RAM (memória de leitura e escrita)**

São chips de memória que podem ser gravados pela CPU a qualquer instante. A CPU usa a RAM para armazenar e executar programas vindos do disco, para ler e gravar os dados que estão sendo processados. É uma memória volátil ( quando o computador é desligado, todos os seus dados são apagados). Por esta razão, os dados e programas devem ficar gravados no disco, que é uma memória permanente.

**Memórias não voláteis**

São memórias cujas informações mantidas não são perdidas caso o computador seja desligado. Exemplo: BIOS (basic input-output system – sistema básico de entrada e saída). Está gravado em uma memória permanente localizada na placa mãe. Tipos de memórias permanentes:

**ROM**

São chips que podem ser lidos pela CPU a qualquer instante, mas não podem ser gravados pela CPU. A gravação é feita pelo fabricante. Este tipo de memória foi usada para armazenar a BIOS.

**PROM**

É uma ROM programável. A gravação pode ser feita apenas uma vez, pois utiliza um processo irreversível.

**EPROM**

É uma ROM programável e apagável. Pode ser programada comportando-se com o uma ROM. A EPROM pode ser apagada com raios ultravioletas de alta potência.

**EEPROM**

É um tipo de memória ROM mais flexível. Pode ser apagada sob controle de software. Utilizada para armazenar as BIOS atuais.

**Memória fora da placa mãe**

A placa mãe contém quase toda a memória de um microcomputador, mas outras placas também podem conter memórias, do tipo RAM e do tipo ROM. As placas de vídeo contém uma ROM com a sua própria BIOS e uma RAM chamada de memória de vídeo

**Placas controladoras**

SCSI (small compact system interface) permite a conexão de diversos periféricos;

IDE (intelligent drive electronics).

**A memória cache**

É uma área reservada de memória que possui duas funções:

* Aumentar o desempenho do computador
* Aumentar o tempo de vida das unidades de disco

Temos dois tipos de memória cache:

* A que vem incorporada à máquina, dessa forma é mais rápida que a memória RAM;
* A que é implementada via software na memória RAM, aumentando o desempenho do acesso ao disco.

**Barramentos**

Um barramento ou bus, é um caminho comum pelo qual os dados trafegam dentro do computador. Este caminho é usado para comunicação e pode ser estabelecido entre dois ou mais elementos do computador.

O tamanho do barramento determina quantos dados podem ser transferidos em uma única vez (16 bits, 32bits, ...)

Um PC possui muitos barramentos, que incluem:

**Barramento do processador**

É o barramento que o chipset (chips de suporte adjacentes contidos na placa mãe) usa para enviar/receber informações do processador.

**Barramento de cache**

É um barramento dedicado para acessar a cache. Usado pelos Pentium Pro e Pentium III.

**Barramento de memória**

Conecta o sub-sistema da memória ao chipset e ao processador.

**Barramento local de entrada/saída (E/S)**

Usado para conectar periféricos de alto desempenho à memória, chipset e processador. Exemplo: placas de vídeo, interface de redes de alta velocidade. Os mais comuns:

* Vesa local bus (VLB)
* Peripheral component inter connect bus (PCI)

**Barramento padrão de entrada/ saída(E/S):**

Usado para periféricos lentos (mouses, placas de som) e também para compatibilidade com dispositivos antigos.

Todos os barramentos possuem duas partes: um barramento de endereçamento (que transfere a informação de onde o dado se encontra) e um barramento de dados (que transfere os dados em si, ou seja, o valor de memória).

**Dispositivos de Entrada e Saída ou Periféricos**

São equipamentos utilizados como portadores das informações que o computador irá processar. Através desses dispositivos, o computador pode armazenar, ler, transferir e receber dados.

Dispositivos de entrada:

Teclado - Mouse - Drive de CD-ROM - Microfone - Scanner

Dispositivos de saída:

Vídeo - Impressora - Alto-falante

Dispositivos de entrada e saída:

Disco rígido - Drive de disquete - Unidade de fita magnética - Modem

**Tipos de comunicação com os dispositivos**

A CPU comunica-se com os periféricos através de circuitos chamados interfaces ou portas de E/S, que implementam a transmissão de dados segundo duas políticas:

* comunicação paralela: impressora;
* comunicação serial: mouse, modem.

**Parte 2 -** [**Estrutura Sequencial - Variáveis, Comandos Básicos e Operadores Aritméticos**](https://sites.google.com/faculdadeimpacta.com.br/lo-p2?pli=1&authuser=2) **-** [**Video**](https://www.youtubeeducation.com/watch?v=cr0xtsc85PI)

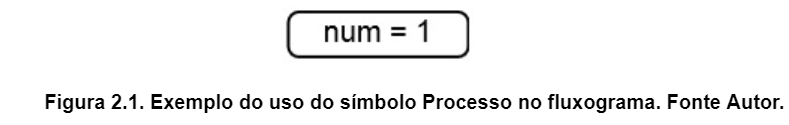
**Resumo**

Para elaborar um programa precisamos entender algumas ferramentas básicas. Como vimos na aula 1, um software recebe informações de dispositivos de entrada, armazena na memória, efetua um processamento (cálculo) com as informações que estão na memória, armazena o resultado na memória e por fim envia o resultado desse cálculo que está na memória para um dispositivo de saída. São esses os componentes que iremos ver nesse texto: Variável (memória) e os tipos de informações que são armazenadas na memória, Comandos de Entrada e Saída, processamento (atribuição =). Por fim, como embasamento para se fazer processamento vamos detalhar melhor os operadores aritméticos.

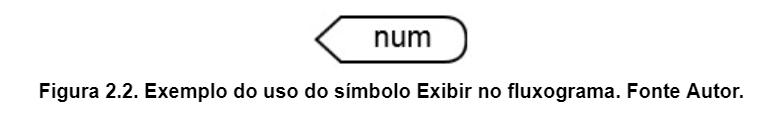
Variáveis e Identificadores

Uma variável tem um nome (identificador) que está associado a um espaço em memória que armazena um valor. Uma variável pode receber diferentes valores durante a execução de um programa, por isso o nome “variável”.

Para atribuir um valor a uma variável, utiliza-se o símbolo de Processo e o operador de atribuição “=”.



Para mostrar um determinado valor na tela, usamos o símbolo de Exibir.

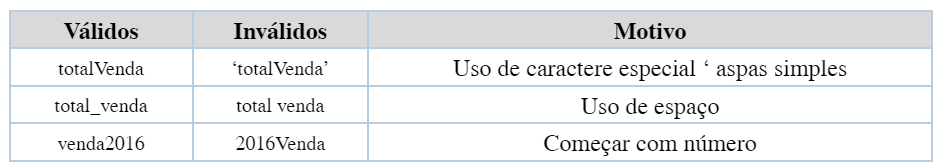


**Como nomear as variáveis**

Um nome ou identificador de uma variável é formado por uma sequência de um ou mais caracteres. Como exemplo, as regras adotadas pelo Python são:

* Pode conter apenas letras, dígitos e underscores (sublinhado);
* Pode começar por uma letra ou underscore;
* Não é permitido o uso de outros caracteres especiais (por exemplo, espaço);
* Não é permitido o uso de palavras reservadas da linguagem;

O identificador deve ser conciso, porém descritivo (idade é melhor que i, tamanho\_nome é melhor que tamanho\_do\_nome\_da\_pessoa).



**Tipos de dados**

**Números**

Um dado numérico é composto por uma sequência de dígitos (0 a 9), um sinal opcional (+ ou –) e um possível ponto decimal (usa-se o ponto e não a vírgula para separar a parte inteira da fracionária). São classificados como inteiro ou de ponto flutuante.

**Texto - letras**

Um texto (string) representa uma sequência de caracteres (letras, dígitos, símbolos especiais). As strings podem ser delimitadas por um par de aspas simples ( ’ ) ou duplas ( ” ).

Exemplos:

“olá”

‘Faculdade Impacta de Tecnologia’

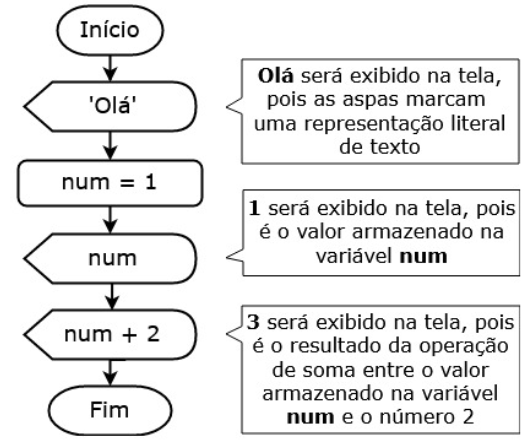
‘Avenida Rudge, 315’

Os dados para serem armazenados na memória devem ser convertidos para números. Com a finalidade de padronizar essa conversão de letras e símbolos para informações em formato numérico para ser armazenada foi criada a tabela ASCII. Veja que por exemplo a letra A (a maiúsculo) é armazenada na memória como o número 65.

**Entrada e saída de dados**

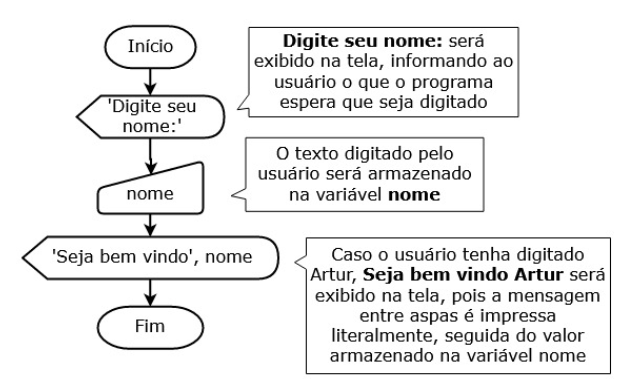
Os programas que iremos escrever poderão obter e apresentar dados aos usuários. Para isso, já vimos em exemplos anteriores o símbolo Exibir (saída de dados).

É possível com isso exibir na tela uma string, o conteúdo de uma variável, o resultado de uma expressão matemática ou booleana, etc.



Se quisermos pedir ao usuário que forneça um valor, via teclado, usaremos o símbolo de Entrada Manual.



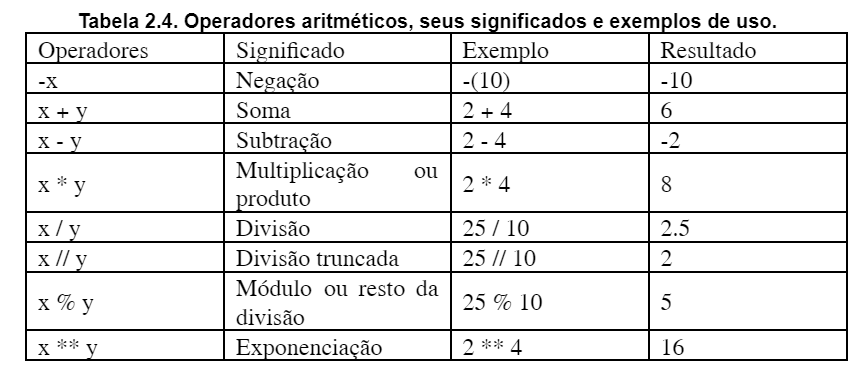


**Operadores e Operadores Aritméticos**

Um operador é um símbolo que representa a operação que pode ser realizada em um ou mais operandos. Operadores que atuam sobre um operando são chamados operadores unários, e operadores que atuam sobre dois operandos são chamados de operadores binários.

**Operadores Aritméticos**

Operadores aritméticos são aqueles cujos operandos são dados de tipo numérico (inteiros ou em ponto flutuante). A tabela a seguir lista os operadores básicos de acordo com a sintaxe do Python, e de maneira geral, a maioria das linguagens de programação tem operadores equivalentes (ou funções que os substituem).



**Expressões e tipos de dados**

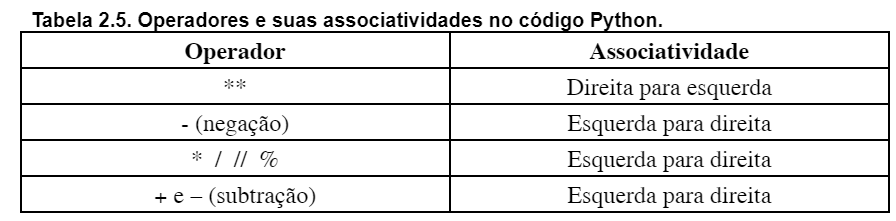
Uma expressão é uma combinação de símbolos (operandos e operadores) que serão avaliados. Assim como na matemática, em programação uma expressão pode conter variáveis, como por exemplo: 4 + 3\*k.

As expressões que avaliam valores do tipo numérico são chamadas de expressões aritméticas.

Para avaliarmos corretamente as expressões acima, é importante sabermos em que ordem devemos realizar as operações, ou seja, é necessário conhecer a precedência dos operadores. O natural seria considerar a precedência da matemática, porém cada linguagem de programação tem sua própria regra, que pode ser igual ou não à da matemática.

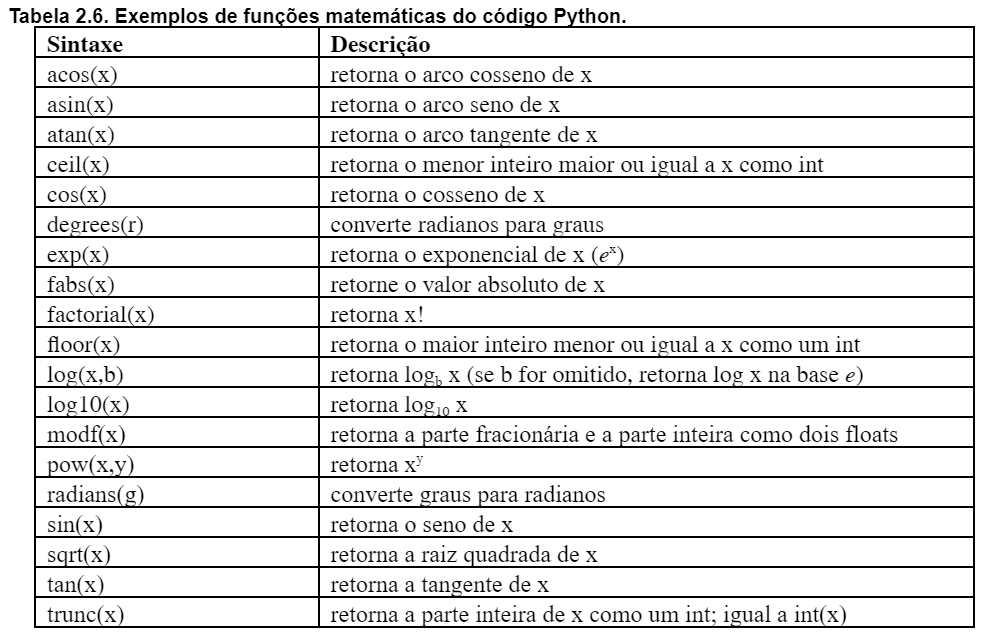
Além da precedência dos operadores, devemos conhecer também como cada linguagem trata os operadores que tenham o mesmo nível de prioridade. À ordem em que isso ocorre damos o nome de associatividade dos operadores.

A tabela a seguir traz a associatividade adotada pelo Python:



**Funções Matemáticas**

Podemos usar um conjunto variado de funções como as trigonométricas (ex: sin, cos, tan, etc) e logarítmicas (ex: log) e também constantes matemáticas (ex: pi). Esse conjunto de funções pode variar dependendo da linguagem de programação utilizada. No nosso curso, utilizaremos as funções disponíveis na biblioteca de funções matemáticas da linguagem Python.

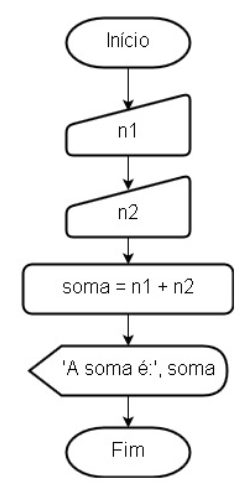


**Estrutura Sequencial**

Nos exercícios construídos anteriormente, sempre foi utilizada a mesma sequência de instruções: receber dados iniciais, realizar operações matemáticas e apresentar resultados finais.

Esta estrutura de controle, caracterizada pela execução das instruções na ordem em que foram escritas, é denominada de Estrutura Sequencial.

Podemos observar esta estrutura no exemplo já trabalhado nas aulas anteriores em que construímos um fluxograma que exibe a soma de dois números informados pelo usuário.



**Exemplo de Fluxograma**

Desenhar um fluxograma que calcula e apresenta o dobro, o triplo e o quadrado de um número digitado pelo usuário.

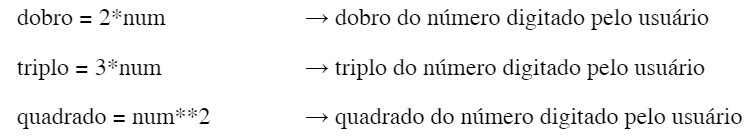
Solução:

Precisamos inicialmente entender o objetivo do exercício. Uma forma interessante de saber onde devemos chegar é identificar no enunciado do exercício o resultado final que está sendo solicitado (saída), os valores que devemos fornecer para conseguirmos fazer o cálculo (entradas) e finalmente elaborar o cálculo necessário (processamento).

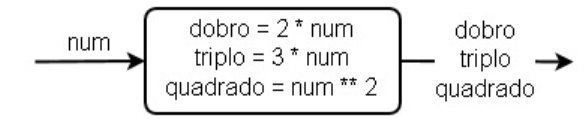
Podemos usar algumas palavras-chave para identificarmos a saída (exibir, apresentar, imprimir). Do texto podemos observar que temos 3 saídas (apresentar dobro, triplo e quadrado). Vamos guardar essas informações em variáveis e chamá-las de dobro, triplo e quadrado.

As entradas também podem ser identificadas por palavras-chave como (dado, digitado, informado, obtido). Nesse caso temos como entrada um único número digitado pelo usuário. Vamos atribuí-lo à variável num.

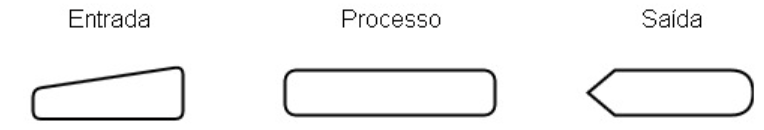
Agora precisamos identificar qual o cálculo que devemos fazer com a entrada (num) para obtermos as saídas pedidas (dobro, triplo e quadrado). Seguem os cálculos:



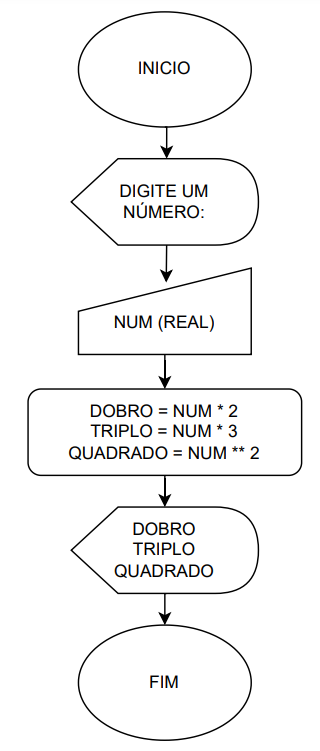
Em seguida, podemos desenhar o seguinte diagrama de entrada/processamento/saída:



E fazendo uso dos seguintes símbolos:

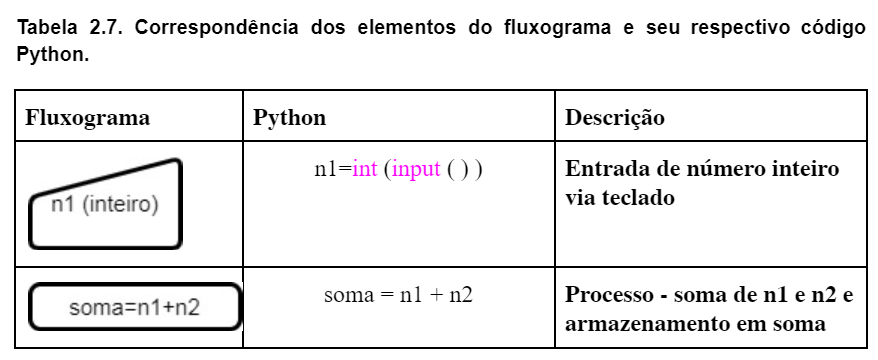


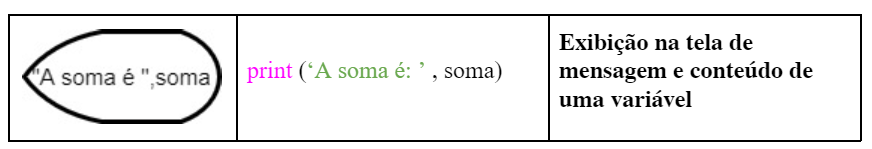
Podemos finalmente desenhar o fluxograma:



Exemplo: Fluxograma e Conversão em Python.

Nesse exemplo, vamos criar um fluxograma para mostrar a soma entre dois números inteiros. Esses números serão digitados pelo usuário e iremos armazená-los nas variáveis n1 e n2. Em seguida, vamos realizar o processo de soma, que consiste em recuperar os valores salvos, somá-los e atribuir o resultado à variável soma. Ao final, este resultado deverá ser apresentado ao usuário.





Faça um fluxograma que:

Defina 3 variáveis com valores que armazenem:

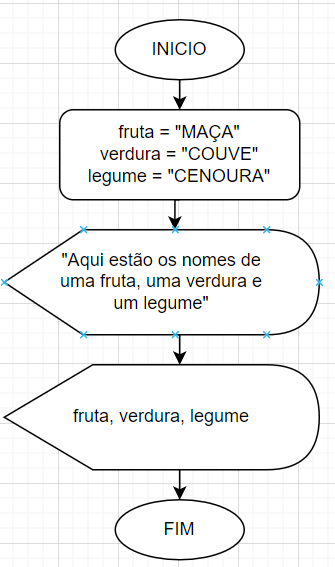
O nome de uma fruta;

O nome de verdura; e

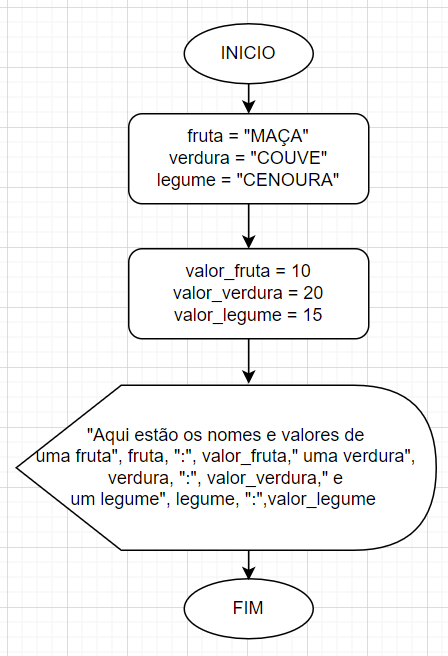
O nome de um legume.

Mostre uma mensagem “Aqui estão os nomes de uma fruta, uma verdura e um legume:”;

Mostre, após essa mensagem, o nome da fruta, da verdura e do legume, respectivamente.



Altere o fluxograma anterior para conter mais três variáveis, com o preço unitário da fruta, verdura e legume. Depois, mostre o preço de cada produto após o nome (nome ao lado do preço, por exemplo: “Maçã: 10”).



**Parte 3 -** [**Como resolver problemas computacionais**](https://sites.google.com/faculdadeimpacta.com.br/lo-p3?pli=1&authuser=2) **-** [**Video**](https://www.youtubeeducation.com/watch?v=v5N9-8Sfpa0)

**Resumo**

Neste texto estão expostas considerações sobre uma técnica simplificada para resolver problemas computacionais salientando os principais pontos: Entendimento, Projeto, Implementação e Testes. Para cada um desses pontos são expostos seus conceitos e exemplificados com vários casos.

**O processo de Resolução de Problemas Computacionais**

Muitos acreditam que para desenvolver um programa basta programar em alguma linguagem. Na verdade, a programação ou construção do programa é apenas uma das etapas desse processo. Antes de um programa ser construído existem passos importantíssimos que se não forem feitos podem levar a um grande prejuízo em termos de tempo e investimentos. Antes da construção precisamos pensar no projeto do programa e antes do projeto precisamos deixar bem claro o que precisa ser feito. Após o programa ser escrito, ele precisa ser testado para verificar o que atende ao que foi proposto. (DIERBACH, 2012)

**As Etapas da Resolução de Problemas Computacionais**

Podemos dividir em quatro as etapas Técnica de Solução de Problemas Computacionais:

1 - Análise e Entendimento

2 - Projeto

3 - Construção ou Implementação

4 - Testes

**Análise e Entendimento**

Esta é a etapa mais importante. Aqui definimos claramente o que deve ser feito, onde devemos chegar, qual o objetivo do programa.

Para exemplificar as Etapas, vamos utilizar o seguinte problema:

Desenhar um fluxograma que receba dois números, calcule e exiba a soma desses números.

Perceba que muitas vezes nos deparamos com um problema incompleto ou mal definido. A primeira e mais importante questão no entendimento é com relação ao objetivo do problema. Está claro o objetivo desse problema? No final das contas, qual a necessidade que esse programa deverá atender?

Para deixar mais claro o entendimento vamos modelar o processo envolvido nesse caso separando as saídas, as entradas e o processamento:

**Saídas:** a soma de dois números. Essa é a informação mais importante. É o nosso objetivo. Uma dica para ajudar a encontrar essa informação, é procurar no texto palavras que tenham a ideia de saída como: exibir, apresentar, mostrar, etc.

**Entradas:** dois números. São os dados que precisamos usar como matéria-prima para conseguir atingir nossa saída, nosso objetivo. Uma dica para ajudar a encontrar essa informação, é procurar no texto palavras que tenham a ideia de entrada como: receba, digite, dados, etc.

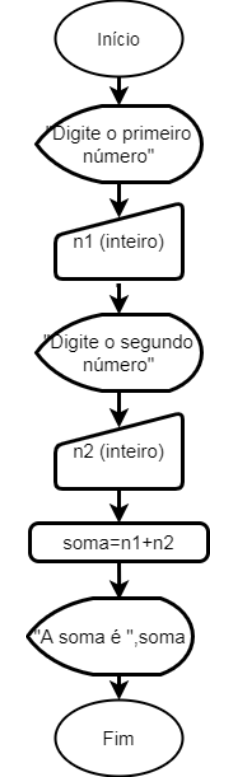
**Processamento:** calcule a soma. O processamento é o que temos que fazer com as entradas para produzir as saídas. Uma dica para ajudar a encontrar essa informação, é procurar no texto verbos de ação envolvendo as entradas e saídas como: calcular, resultar, somar (aqui entra qualquer verbo que indica cálculo ou processamento), etc.

**Projeto**

Nesta etapa, devemos achar uma estratégia de solução de maneira a conseguir efetuar o processamento. Estamos procurando aqui um algoritmo que consiga transformar as entradas nas saídas desejadas. Devemos lembrar que um algoritmo além de envolver processos, passo-a-passo de uma solução também pode envolver uma determinada forma de armazenar os dados para conseguir efetuar o passo-a-passo. No nosso exemplo, esse algoritmo é uma simples soma.

**Implementação ou Construção**

Nesta etapa devemos definir qual ferramenta/linguagem será usada para a construção do algoritmo definido na etapa de projeto. Aqui aplicamos as ferramentas necessárias de lógica de programação para implementar o algoritmo. Essa é a etapa comumente conhecida como “programar”. No nosso exemplo, utilizaremos o fluxograma:



Perceba que a utilização de um fluxograma permite que posteriormente ele possa ser convertido para qualquer linguagem que se queira. O Fluxograma carrega com ele toda a inteligência do entendimento e projeto de software.

**Testes**

Essa é uma etapa costumeiramente esquecida por muitos programadores. Não menos importante que as outras, essa etapa bem feita define a qualidade do programa. Tanto no quesito mais importante de atender ao que foi proposto atingindo seu objetivo com precisão, quanto na eficiência da maneira como foi construído. A disciplina de testes é muito abrangente, vamos mostrar aqui apenas algumas das muitas técnicas de elaboração de testes.

**Como testar um programa**

**Teste da caixa-preta**

Existem formas de testar a corretude de programas. Uma das formas é o teste de caixa-preta, cujos passos resumidamente são:

1 - Escolhe-se uma entrada cuja saída correta correspondente seja conhecida, sem precisar usar o programa que será testado;

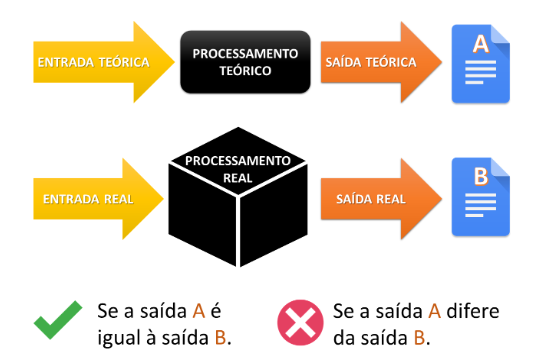
2 - Simula-se a execução do fluxograma usando a entrada escolhida (TESTE DE MESA);

3 - Compara-se a saída simulada com aquela teórica inicialmente esperada;

4 - Caso haja diferença entre a saída teórica e a saída real, muito provavelmente o programa está incorreto e necessita de correções. As devidas modificações são feitas e retorna-se ao passo (b);

5 - Caso não existam divergências entre as saídas, opta-se por: (I) encerrar os testes ou (II) submeter o programa a outros casos de teste.

Note que testes de caixa-preta não consideram a estrutura interna do programa, ou seja, o algoritmo usado para construir o programa, assim como a linguagem de programação em que foi implementado, não importam.



Lembre-se, testes não garantem, necessariamente, que um programa está correto. Um programa que seja aprovado em todos os casos de teste aos quais foi submetido não está obrigatoriamente correto, pois pode haver um caso de teste ausente que geraria uma falha.

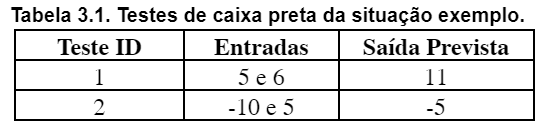
Para garantir a corretude de um programa por meio de testes é necessário fazer um teste exaustivo. Testes exaustivos submetem o programa a todas as entradas esperadas possíveis, o que muitas vezes é impraticável. Imagine quantas possíveis entradas esperadas existem para um programa que soma dois números inteiros e exibe o resultado. Infinitas!

Uma forma de garantir a corretude de um programa sem precisar construir testes, é por meio de prova formal, porém esse método exige maiores conhecimentos matemáticos, técnicas avançadas de análise de algoritmos e criatividade. Provas formais costumeiramente demandam mais recursos financeiros e tempo para serem satisfatoriamente concluídas em programas mais extensos.

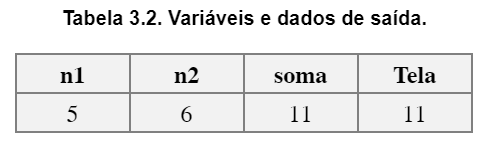
Geralmente, para sistemas não críticos (aqueles que não gerenciam laboratórios com doenças altamente contagiosas; não administram usinas nucleares; não controlam produção de foguetes; não automatizam cirurgias, aviões, trens, etc.) são montadas sequências de testes com boa abrangência, testando principalmente as extremidades das possíveis entradas esperadas. Isso amplia o nível de confiança nas soluções propostas. Em nossas aulas optamos por essa abordagem simplificada para o teste de programas.

O uso de testes de mesa pode auxiliar na execução de testes de caixa-branca, pois permite que toda instrução que implique em mudança nas variáveis seja representada em uma linha-coluna da tabela. No caso de testes de mesa com estruturas condicionais, é facilitador acrescentar uma coluna com a expressão da condição de seleção, sendo útil para perceber erros na definição da condição (frequentes com iniciantes).

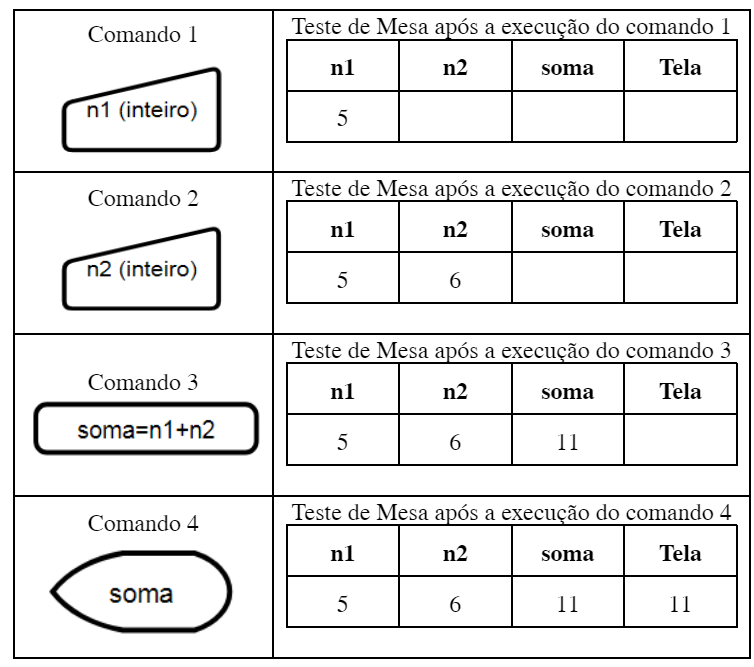
Vamos aplicar a técnica da caixa preta ao exercício da soma de dois números, Vamos pensar em dois testes:



Para a elaboração do teste de mesa, criamos uma tabela com as variáveis usadas e com um coluna (Tela) que mostra o que está sendo exibido na tela:



Em seguida, a cada comando executado no fluxograma, registramos o valor que as variáveis envolvidas vão recebendo, conforme mostrado na tabela a seguir, na qual executamos o teste 1:



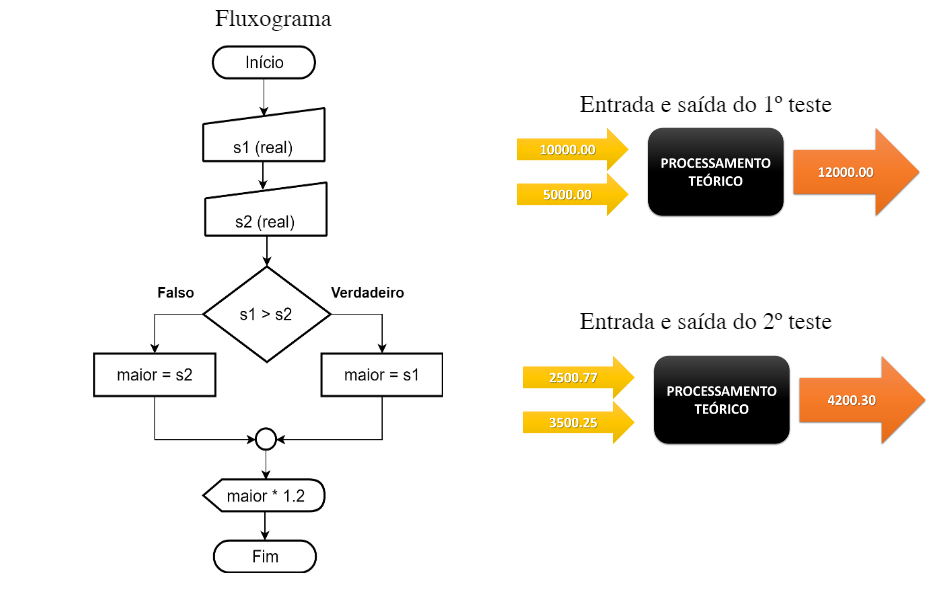
**Teste da caixa-branca**

Também é possível usar testes de caixa-branca, em que a estrutura interna do programa é avaliada, ou seja, cada instrução do algoritmo implementado em uma linguagem de programação será analisada de acordo com a sequência de execução. Neste formato de teste é necessário "abrir a caixa e ver o que está dentro".

O uso de testes de mesa pode auxiliar na execução de testes de caixa-branca, pois permite que toda instrução que implique em mudança nas variáveis seja representada em uma linha-coluna da tabela. No caso de testes de mesa com estruturas condicionais, é facilitador acrescentar uma coluna com a expressão da condição de seleção, sendo útil para perceber erros na definição da condição (frequentes com iniciantes).

Bons testes de caixa-branca para programas com estruturas de seleção buscam passar por todos os caminhos do algoritmo, garantindo que sejam executadas e analisadas tanto as instruções do "bloco do caminho se verdadeiro" quanto do "bloco do caminho se falso", se houver.

Pense no seguinte exemplo: crie um fluxograma que leia como entrada dois salários distintos e exiba uma mensagem com o valor do maior deles, acrescido de 20% de bônus. Vamos construir um fluxograma, fazer dois testes de caixa-preta (sem comparação com as saídas de um programa real, pois não o codificamos) e dois testes de caixa-branca.



Como fazer o teste de mesa? Video: [(#9) Como fazer teste de mesa?](https://youtu.be/Atcfaafvs4M)

**Parte 4 -** [**Estrutura de Seleção - Operadores e Expressões Lógicas**](https://sites.google.com/faculdadeimpacta.com.br/lo-p4?pli=1&authuser=1) **-** [**Video**](https://www.youtubeeducation.com/watch?v=OlG_JnEP1pk)

**Resumo**

Nem todos os comandos de um programa precisam ser sempre executados. Existem situações em que precisamos executar os comandos seletivamente e inclusive às vezes não executar. Vamos aprender nesse texto sobre a ferramenta que permite que isso seja feito - Estrutura de Seleção. Como base para a utilização dessa ferramenta iremos aprender um pouco mais sobre os operadores e as expressões lógicas. Discutimos também os conceitos de estrutura de seleção simples e composta. Tanto a estrutura de seleção simples quanto a estrutura de seleção composta utilizam o símbolo de decisão. Através dessa decisão definimos obrigatoriamente dois caminhos no nosso fluxograma. Abaixo são expostos os conceitos dessas estruturas assim como exemplos de utilização. Ao final é mostrado um exemplo usando fluxograma com sua conversão para Python.

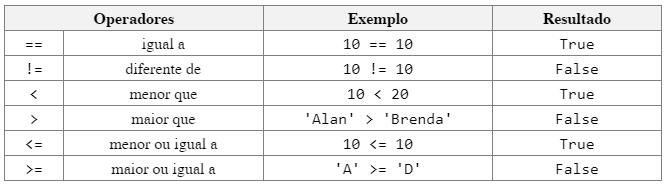
**Expressões lógicas**

Como vimos anteriormente, o resultado de expressões aritméticas são números inteiros ou reais. Em expressões lógicas, o resultado gerado é um valor lógico (também conhecido como booleano), ou seja, Verdadeiro (True) ou Falso (False).

Novamente, como analogia, em expressões aritméticas os operandos costumam ser números e os operadores são aritméticos (adição, subtração, multiplicação, divisão e etc.), já com expressões lógicas os operandos também podem ser números, porém há presença de operadores relacionais e/ou operadores lógicos.

**Operadores relacionais**

Os operadores relacionais são usados em comparações entre valores. O resultado da comparação será um valor lógico, como exposto no quadro a seguir:

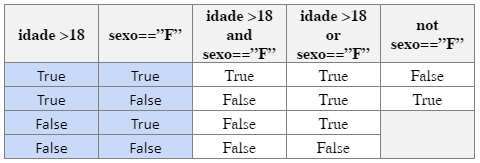


**Operadores lógicos**

Os operadores lógicos podem ser usados para construir expressões lógicas mais complexas, permitindo a combinação de comparações. Consideraremos três operadores: E (and), OU (or) e NÃO (not).

* O operador lógico and resulta verdadeiro somente se os dois operandos forem verdadeiros.
* O operador lógico or resulta verdadeiro se ao menos um dos dois operandos for verdadeiro.
* O operador lógico not resulta verdadeiro se o operando for falso e resulta em falso caso seja verdadeiro.

Veja a tabela verdade dos operadores lógicos:



É preciso ser cauteloso ao usar operadores lógicos. Por exemplo, na matemática, para indicar que um valor está dentro de um determinado intervalo escrevemos:

(1 <= num <= 10)

No entanto, na maioria das linguagens de programação esta expressão não faz sentido. Para entender porque, vamos assumir que a variável num tem o valor 15. A conversão da expressão anterior para uma vasta gama de linguagens de programação seria assim:

(1 <= num <= 10) → (1 <= 15 <= 10) → (True <= 10) → ???

Em diversas linguagens de programação a expressão será avaliada de modo diferente ao tradicionalmente aceito na matemática. Inicialmente se avaliaria a primeira parte da expressão lógica (1 <= num) que geraria um resultado booleano True; na sequência seria avaliada a segunda parte da expressão (True <= 10). Porém, não faz sentido verificar se True é menor ou igual a 10. A forma comumente aceita para a expressão anterior usaria o operador and:

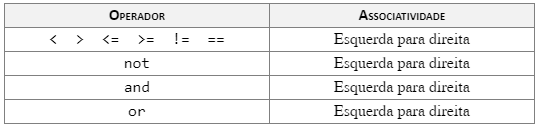
(1 <= num) and (num <= 10)

ou

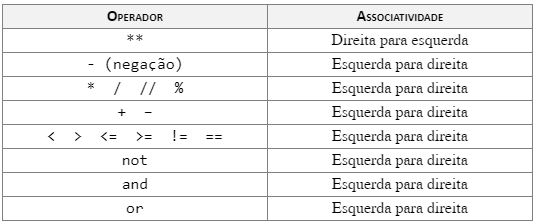
(num >= 1) and (num <= 10)

**Precedência de operadores (prioridade na avaliação)**

A precedência dos operadores relacionais e lógicos, bem como suas associatividades, é definida conforme explicitado na seguinte tabela:

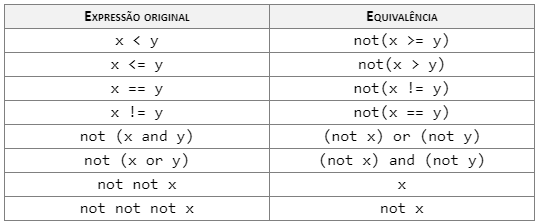


Expressões lógicas também podem conter operadores aritméticos. Agora podemos expandir nossa tabela de operadores conhecidos e suas respectivas precedências:



**Expressões lógicas equivalentes**

Assim como há expressões aritméticas equivalentes, como a\*(b+c) é igual a a\*b + a\*c, também existem expressões lógicas equivalentes. É importante reconhecer as equivalências para simplificar expressões e outras finalidades relacionadas.



**Estruturas de seleção (estruturas condicionais)**

A partir de agora vamos estudar as estruturas condicionais que permitem que programas executem sequências diferentes de instruções, dependendo da avaliação de uma expressão lógica.

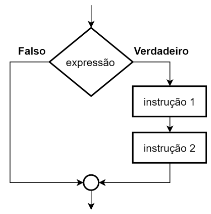
Em programação, o uso de condições para permitir a escolha de executar ou não um trecho de código é muito utilizado, principalmente quando precisamos incluir no programa condições de controle, para evitar situações não permitidas que podem resultar em erros. Por exemplo, para evitar divisões por zero.

Quando há apenas um bloco de instruções cuja execução está condicionada ao resultado de uma expressão lógica, usamos uma estrutura de seleção simples.

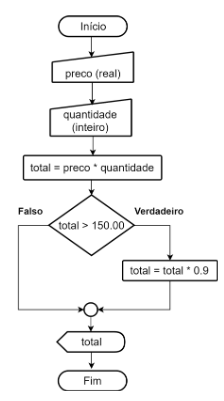
Existe outra estrutura, chamada de estrutura de seleção composta, em que ainda temos apenas uma condição a ser avaliada, porém dois caminhos que podem ser seguidos. Um dos caminhos leva à execução do bloco de instruções condicionado ao resultado verdadeiro e o outro à execução do bloco de instruções condicionado ao resultado falso.

**Estrutura de seleção simples**

Na estrutura de seleção simples ou estrutura condicional simples, um bloco de código só será executado caso a expressão da condição da estrutura resulte em um valor lógico verdadeiro. Caso a condição resulte falsa, o bloco será ignorado e as instruções posteriores, externas à estrutura condicional, serão executadas.



Suponha a seguinte situação: dado o preço de um produto comercializado por uma loja e a quantidade comprada por um cliente, exiba o valor total da compra. Considere que se o total for maior do que R$ 150,00 haverá desconto de 10%. Um fluxograma para esta situação pode ser representado como a seguir:

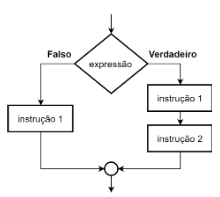


Note que o valor da variável total sempre será exibido, independentemente se houver ou não desconto, porém se o valor de total for superior a 150.00 o trecho de código que o reduz em 10% será executado, pois a condição da seleção resultará verdadeiro.

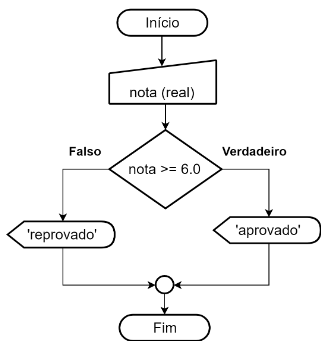
**Estrutura de seleção composta**

Na estrutura de seleção composta ou estrutura condicional composta, há dois caminhos possíveis, cada um com um conjunto de instruções, mas apenas um deles será executado.

Um bloco de código só será executado caso a condição da estrutura de seleção resulte em um valor lógico verdadeiro. Caso a condição resulte falso, outro bloco de códigos será executado.



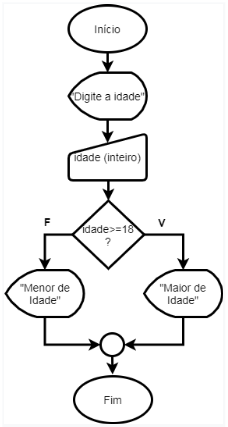
Suponha a seguinte situação: dada a nota de um aluno, exibir 'aprovado' caso esta seja maior ou igual a 6.0, e 'reprovado' caso contrário. Um fluxograma para esta situação pode ser representado assim:



Note que neste caso somente uma mensagem será exibida. Se a condição da seleção resultar verdadeiro, será exibido 'aprovado', caso contrário será exibido 'reprovado'. Costumeiramente, chamamos o lado que é executado quando a condição resulta falso de "senão".

**Exemplo 1:**

Elaborar um Fluxograma que recebe uma idade e exibe a mensagem Maior de Idade, caso a idade seja maior ou igual a 18 e exiba a mensagem Menor de Idade, caso contrário.



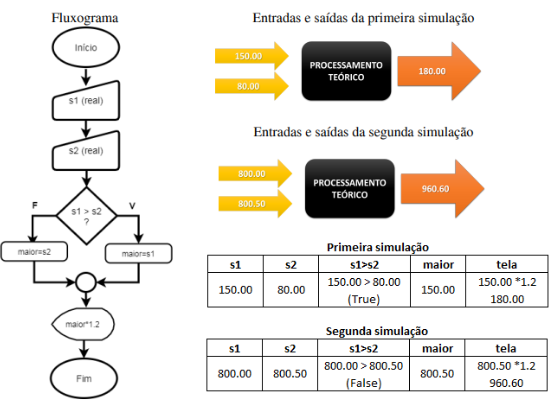
**Saídas:**

Nesse exemplo, veja que a saída são duas mensagens: Maior de Idade (caso a idade seja maior ou igual a 18) e menor de idade (caso a idade seja menor que 18).

**Entradas:**

Temos uma entrada - idade, que é um número inteiro.

Na criação do fluxograma, percebemos que primeiro fizemos a entrada de dados - idade e após isso utilizamos uma estrutura de seleção justamente para definir qual das duas mensagens será exibida dependendo do resultado do teste lógico envolvendo a idade (idade>=18). Caso o teste seja verdadeiro, significa que a idade digitada pelo usuário é maior ou igual a 18. Sendo assim, seguimos com o caminho da direita e será exibida a mensagem “Maior de idade”. Caso o resultado do teste seja falso, será exibida a mensagem “Menor de idade”. Perceba que quando usamos uma estrutura de seleção nunca serão usados os dois caminhos simultaneamente. Se usamos o caminho do verdadeiro, o caminho do falso não é usado e vice-versa.



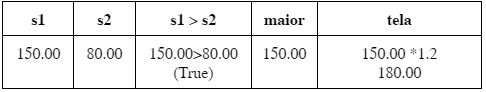
Nesse exemplo temos 2 entradas (os dois salários) que foram armazenadas em duas variáveis, s1 e s2. Criamos uma terceira variável com o objetivo de armazenar o maior salário chamada maior.

Caso o primeiro salário seja maior que o segundo (s1>s2), a variável maior recebe o valor do primeiro salário (maior=s1). Receberá o segundo salário caso esse teste s1>s2 seja falso - (maior=s2). Ao final, após a estrutura de seleção, é apresentado o valor da variável maior multiplicado por 1.2 (aumento de 20%) na tela.



No diagrama acima estamos mostrando também dois testes e seus resultados teóricos.Veja nos teste abaixo temos como entradas 150.00 e 80.00 e veja que na saída é apresentado o valor 180.00 (150\*1.2).

Podemos seguir a simulação desse teste pelo quadro abaixo:



Nesse quadro notamos que são mostrados em colunas as variáveis s1,s2 e maior, a expressão lógica da decisão s1>s2 e o que é exibido na tela.

A simulação ocorre de cima para baixo. Então, temos:

1 - a variável s1 recebe 150.00 (digitado pelo usuário)

2 - a variável s2 recebe 80.00 (digitado pelo usuário)

3 - é feito um teste s1>s2 (150.00>80.00) resultando em Verdadeiro (o fluxo toma o caminho da direita

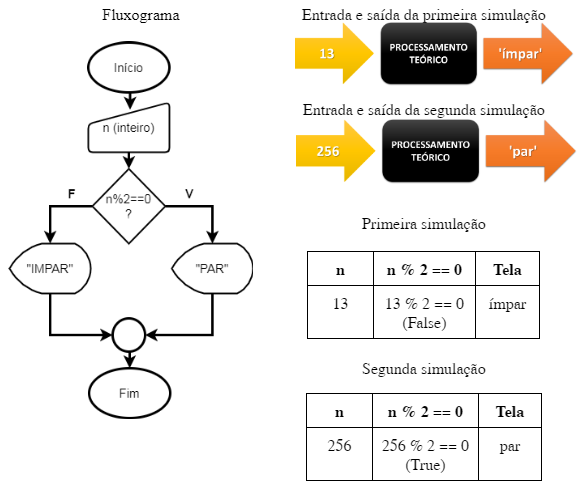
4 - a variável maior recebe 150,00 que é o valor da variável s1

5 - A estrutura de seleção finaliza e é exibo cálculo maior\*1.2 (150.00\*1.2) que apresenta na tela o valor 180.00

O segundo teste com os valores 800.00 e 800.50 ocorre de maneira semelhante. Observe que neste exemplo o objeto de escolha da estrutura de seleção foram comandos de atribuição (cálculos), enquanto que no exemplo anterior foram comandos de saída.

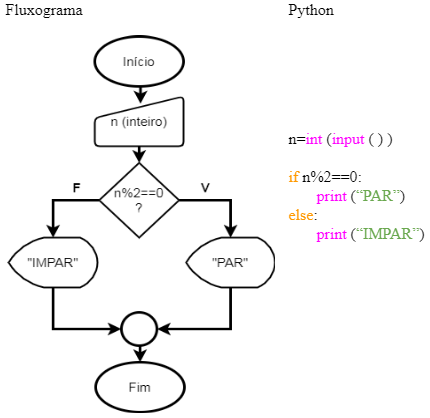
**Exemplo 2**

Vamos elaborar um fluxograma de um programa que recebe um número inteiro e exibe uma mensagem indicando se ele é par ou ímpar (vamos usar o operador % para obter o resto de uma divisão inteira).

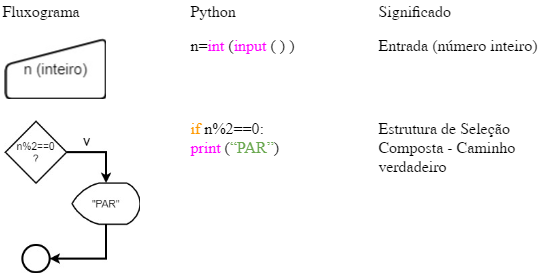


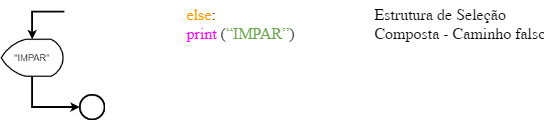
Nesse exemplo, diferentemente aos anteriores, veja que na decisão foi colocado um cálculo (n%2==0). A expressão lógica é avaliada depois do cálculo (n%2). Esse é o resto de n que foi digitado pelo usuário por 2. Quando obtemos o resto da divisão por 2 igual a 0 significa que o número é par. Caso seja 1 é ímpar.

**Conversão para Python**



Veja a equivalência dos comandos:





Estrutura condicional - Fluxograma -[aula09 - Estrutura condicional - Fluxograma](https://youtu.be/kN8Kk1BNrp0)

Fluxograma (estrutura de decisão) + exemplo - <https://youtu.be/PBFeqj1_2tA?si=L3_-vrw1X695TJ1j>

**Parte 5 -** [**Estrutura de Seleção Encadeada**](https://sites.google.com/faculdadeimpacta.com.br/lo-p5?pli=1&authuser=1) **-** [**Video**](https://www.youtubeeducation.com/watch?v=vTWk4RYEy3k)

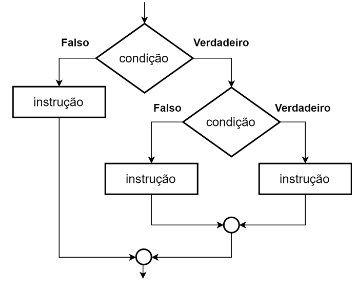
**Resumo**

Neste texto é discutido o conceito de estrutura de seleção encadeada ou aninhada. Como exemplificação mostra-se uma implementação de um exemplo usando fluxogramas, um segundo exemplo usando fluxogramas juntamente com uma simulação. Ao final é mostrado um terceiro exemplo usando fluxograma com sua conversão para Python.

**Estrutura de seleção encadeada**

Tanto na estrutura de seleção simples quanto na seleção composta há apenas uma condição para ser avaliada e que condicionará o fluxo de execução do programa. Portanto, há no máximo uma bifurcação para um caminho, entre dois possíveis.

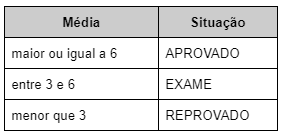
E quando é necessário resolver um problema que possui mais de dois caminhos possíveis? Para lidar com situações como essa, podemos usar uma estrutura de seleção encadeada (também conhecida como estrutura de seleção aninhada).

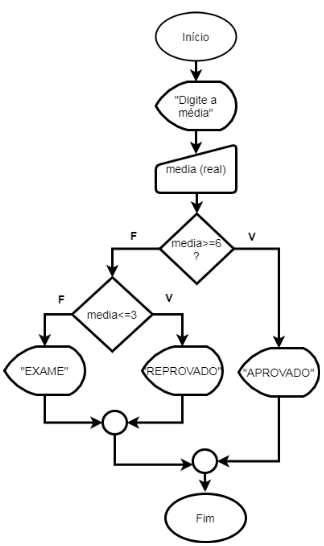


A estrutura de seleção encadeada ocorre quando uma seleção tem como ação uma outra seleção ou quando temos mais de duas possibilidades para uma decisão. A seleção encadeada ou aninhada é o agrupamento de uma ou várias seleções internas a uma seleção externa.

**Exemplo 1:**

Desenhar um fluxograma que obtém a média de um aluno e exibe a sua situação segundo a tabela:





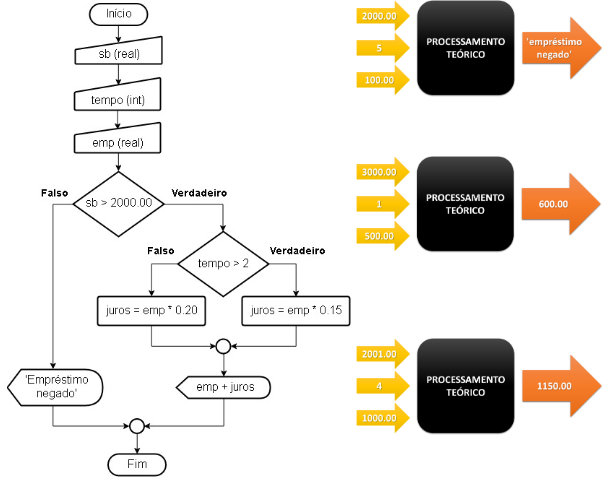
Perceba que nesse fluxograma temos 3 possibilidades de caminhos a serem percorridos, uma para cada status APROVADO, REPROVADO e EXAME. Na primeira decisão media >=6 definimos dois caminhos. No caminho da direita (verdadeiro), temos apenas uma possibilidade (media>=6 portanto aluno aprovado). Já no caminho da esquerda (falso), temos duas possibilidades: média <=3, ou seja, reprovado e média entre 3 e 6 (como ele caiu no caminho da esquerda, o valor da média é menor que 6). Precisamos, portanto, de mais de uma estrutura de seleção para que possamos decidir em qual dessas duas possibilidades a média testada se encontra. Perceba, ainda, que essa outra estrutura de seleção está dentro da primeira (no ramo esquerdo dela) e, portanto, é uma estrutura encadeada ou aninhada (ninho).

**Exemplo 2:**

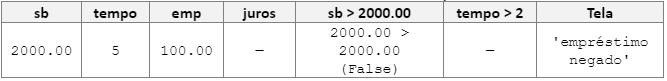
Considere a seguinte situação: um banco oferece uma modalidade de empréstimo de parcela única para qualquer cliente cujo salário bruto ultrapasse R$ 2.000,00. Essa modalidade também considera que, caso o cliente tenha mais de dois anos de contrato, serão cobrados juros de 15% sobre o valor emprestado, caso contrário os juros serão de 20%. Se o salário bruto não atingir o mínimo estipulado, o empréstimo será negado. O gerente do banco quer um programa que dados como entrada o salário bruto (em reais e centavos), o tempo de contrato do cliente com o banco (em anos) e o valor solicitado de empréstimo (em reais e centavos), exibe o valor da dívida do cliente, isto é, o valor emprestado acrescido de juros.

Note que com apenas os recursos das estruturas de seleção simples e composta não seria simples modelar esta situação, pois o enunciado evidencia que existem mais de dois possíveis fluxos de execução no algoritmo, que dependem dos dados passados como entrada.

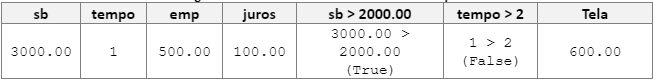
Um fluxograma adequado para esse problema é ilustrado a seguir:



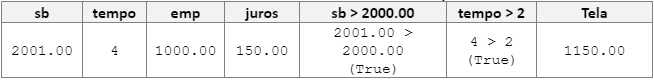
**1º Teste de mesa**



**2º Teste de mesa**



**3º Teste de mesa**

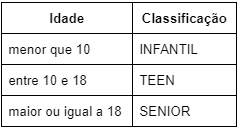


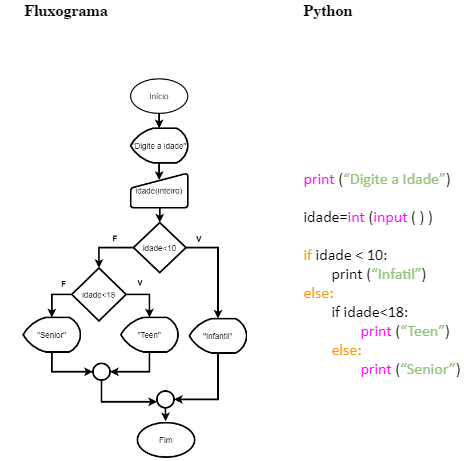
**Comparativo entre fluxograma e Python**

Abaixo podemos identificar um fluxograma e sua conversão para python.

Exemplo 3:

Desenhar um fluxograma que obtém uma idade e exibe uma mensagem segundo a tabela abaixo:





Lógica de Programação - Condicional (ou seleção) encadeada. <https://youtu.be/0k8vpmECZDo>