

# Projeto -Jovem

## Lógica de programação

### SCRATCH



# Sumário

Capítulo 1. Lógica de Computadores.....	1
1.1. Introdução.....	1
1.2. Principais elementos de Hardware.....	1
1.2.1. Memória Principal.....	1
1.2.2. Processador ou Unidade Central de Processamento(CPU).....	2
1.2.3. Unidades de entrada e saída.....	2
1.3. Lógica de programação.....	2
1.3.1. Sequência lógica ( Passos lógicos ).....	2
1.3.2. O que são instruções?.....	2
1.4. Definição de algoritmo.....	3
1.4.1. Exemplo de um algoritmo (Sequência lógica do dia-a-dia).....	3
1.4.2. Fazendo algoritmos - Praticando algoritmos do dia-a-dia.....	4
1.5. Pseudocódigo (Forma genérica de escrever um algoritmo , utilizando uma linguagem simples).....	4
1.6. Regras para construção do algoritmo .....	4
1.6.1. Estrutura principal de um algoritmo .....	4
1.6.2. Fases.....	5
1.7. Tipos de Dados.....	5
1.8. Identificadores.....	5
1.8.1. Formação de identificadores.....	5
1.9. Literais (ou constantes).....	6
1.10. Variáveis.....	6
1.10.1. Declaração de variáveis.....	6
1.11. Atribuição de valores.....	7
1.12. Comando de Entrada e saída.....	7
1.13. Diagrama de Bloco.....	7
1.14. Operadores Relacionais.....	8
1.15. Operadores Lógicos.....	9
1.16. Estruturas de Decisão e Repetição .....	10
1.16.1. Estrutura SE...FAÇA ISSO / SENÃO.....	10
1.16.2. Estrutura Enquanto .....	11
1.16.3. Estrutura PARA <VAR> DE <início> ATÉ <final> FAÇA.....	11
Capítulo 2. Scratch.....	13
2.1. Introdução ao Scratch.....	13
2.2. Ambiente ou Interface de programação do Scratch.....	13
2.2.1. Menu Geral.....	14
2.2.2. Alguns Elementos do Scratch.....	15
2.2.2.1.Área de comando ou área de Script.....	15
2.2.2.2.Palco.....	15
2.2.2.3.Mostrando sistema de coordenadas e orientações.....	16
2.2.2.4.Sprite .....	16
2.2.2.5.Opções de rotação.....	16
2.2.2.6.Informações do objeto.....	16
2.2.2.7.Opções para área de script.....	16
2.2.2.8.Encolher / Crescer / Apagar / Duplicar Objeto.....	17
2.2.2.9.Modos de Apresentação.....	17
2.2.3. Categoria de comandos.....	17
2.3. Tipos de comandos do Scratch.....	17
2.4. Mudando o Sprite(Lista de Sprites).....	18

2.5. Movendo um comando e executando-o .....	18
2.6. Trabalhando com a aba Movimento .....	19
2.6.1. Trabalhando com o mova.....	19
2.6.2. Trabalhando com o vire.....	19
2.6.3. Alterando o numero de graus e passos no comando.....	20
2.6.4. Movendo e virando.....	20
2.6.5. Mostrando como voltar para a posição inicial .....	20
2.6.6. Trabalhando mais com a posição e a orientação.....	21
2.7. Trabalhando em blocos.....	21
2.8. Retirando uma ação de um bloco.....	21
2.9. Removendo uma ação.....	21
2.10. Renomeando um objeto.....	21
2.11. Trabalhando com aba aparência.....	22
2.11.1. Trabalhando com Trajes.....	22
2.11.2. Trabalhando com “Diga” e “Pense”.....	22
2.11.3. Mudando os efeitos .....	22
2.11.4. Modificando os tamanhos.....	23
2.11.5. Aparecendo / desaparecendo.....	23
2.11.6. Trabalhando com camadas.....	23
2.12. Trabalhando com a aba de som (adicionando som).....	23
2.12.1. Trabalhando com o comando Toque o som.....	23
2.12.2. Trabalhando com Toque o tambor e Toque a nota.....	24
2.12.3. Trabalhando com volume e ritmo.....	24
2.13. Ensaaiando um pé de dança.....	24
2.14. Trabalhando com a aba caneta.....	24
2.14.1. Usando o Abaixar, levantar, carimbe e limpe.....	25
2.14.2. Mudando a cor, o tom e o tamanho da caneta.....	25
2.15. Trabalhando com a aba controle.....	25
2.15.1. Usando o quando.....	25
2.15.2. Usando os controles se / se senão.....	25
2.15.3. Uso do controle sempre.....	26
2.15.4. Usando os controles repita / repita até / sempre se.....	26
2.15.5. Usando o pare e espere.....	26
2.16. Iniciando e parando um programa.....	26
2.17. Usando Teclas – Detectando teclas pressionadas.....	27
2.18. Mudando a cor de objeto quando pressionada determinada tecla.....	27
2.19. Mudando os trajes com controle.....	27
2.20. Tocar na borda e voltar.....	28
2.21. Desenhando uma estrada.....	28
2.22. Desenhando um quadrado.....	28
2.23. Trabalhando com a aba Sensores.....	29
2.23.1. Comandos tocando em / tocando na / tocando cor .....	29
2.23.2. Comandos pergunte e resposta.....	29
2.23.3. Comandos mouse x e mouse y .....	29
2.23.4. Comandos mouse pressionado / tecla pressionada / temporizador.....	29
2.24. Movimentando Sprite por Sensores.....	29
2.25. Movendo-se com relação ao temporizador.....	30
2.26. Fala e Calcular.....	30

2.27. Trabalhando com a aba Variáveis.....	30
2.27.1. Usando variáveis.....	31
2.27.2. Listas.....	31
2.28. Caracteres.....	31
2.29. Criando um pequeno aplicativo.....	32
2.30. Abrindo projetos já existentes.....	33
2.31. Compartilhando.....	33
Capítulo 3. Python.....	34
3.1. Introdução ao Python.....	34
3.2. O que é o Python.....	34
3.3. Iniciando o uso do Python.....	35
3.4. Utilizando o Prompt do python.....	35
3.4.1. Utilizando o Python como uma calculadora (utilizando números e operadores aritméticos simples).....	35
3.4.2. Inserindo comentários(#); .....	36
3.4.3. Tipo de variáveis / Tipos de Dados.....	36
Variáveis.....	36
3.4.4. Atribuição de variáveis.....	37
3.4.5. Utilização de constantes e variáveis no Python.....	37
3.4.6. Utilizando Strings .....	38
3.4.7. Visualizando o tipo de variável trabalhada (type()).....	38
3.4.8. Imprimindo resultados (print()).....	39
3.4.9. Utilizando a quebra de linha (\n).....	40
3.5. Trabalhando com bloco de instruções e fazendo um programa.....	40
3.6. Rodando no shell o programa .....	41
3.7. Docstrings(quando precisa-se inserir um comentário maior, como uma documentação);.....	41
3.8. Trabalhando com a biblioteca ou módulo math .....	42
3.9. Mais string:.....	43
3.9.1. Operadores e Strings.....	44
3.9.2. Alinhamento de Strings.....	44
3.10. Entrada de dados (raw_input);.....	46
3.11. Operadores Aritméticos e Expressões Lógicas.....	47
3.12. Operadores Lógicos e Operadores Relacionais .....	48
3.12.1. Trabalhando no shell com operadores lógicos.....	48
3.12.2. Expressões Lógicas.....	49
3.13. Estrutura de Seleção(Decisão ou Condição – if).....	49
3.13.1. Seleção Composta (if..else).....	50
3.13.2. Seleção Encadeada ou Decisão Aninhada.....	50
3.13.3. A construção if..elif..else aninhados.....	51
3.14. Controle de escopo por indentação.....	52
3.15. Criando Funções.....	52
3.15.1. Definindo uma Função.....	52
3.15.2. Escopo de Variáveis.....	53
3.15.3. Argumentos Requeridos.....	54
3.15.4. Argumento Padrão.....	54
3.15.5. Criando e utilizando seu próprio módulo.....	55
3.16. Listas.....	55

3.16.1. Introdução.....	55
3.16.2. Listas.....	55
3.16.3. Acessando os valores de uma Listas.....	56
3.16.4. Operações básicas da Lista.....	56
3.16.5. Atualizando os valores da Lista .....	56
3.16.6. Deletando valores da Lista .....	57
3.17. Estrutura de Repetição .....	57
3.18. Estrutura de repetição FOR;.....	57
3.18.1. Função range().....	58
3.19. Comando de Repetição while .....	59
3.19.1. O Laço Infinito .....	60
3.19.2. A declaração break .....	60
3.19.3. A declaração continue.....	61
3.19.4. A declaração pass.....	61
3.20. Utilizando Arquivos.....	62
3.20.1. A função open().....	62
3.20.2. A função close().....	63
3.21. Escrevendo nos arquivos com a função write().....	64
3.22. Lendo informações do arquivo com a função open() .....	64
3.22.1. Posição do cursor.....	65

# Capítulo 1. Lógica de Computadores

---

## 1.1. Introdução

---

O computador é uma máquina que realiza uma variedade de tarefas de acordo com instruções específicas. É uma máquina de processamento de dados, que recebe dados através de um dispositivo de entrada e o processador os manipula de acordo com um programa.

O computador tem dois componentes principais. O primeiro é o Hardware que é a parte palpável (que se pode tocar) do computador. Ele é composto de partes eletrônicas e mecânicas.

O segundo componente principal é o Software que é a parte impalpável (não se pode tocar) do computador. Ele é composto de dados e dos programas de computador.

O software é um programa que o computador usa para funcionar. Ele é armazenado em algum dispositivo de hardware como um disco rígido, mas é em si mesmo intangível. Os dados que o computador usa podem ser qualquer coisa que o programa precise. Os programas agem como instruções para o processador.

Alguns tipos de programas de computador:

1. Programas de Sistemas: Programas necessários para que o hardware e o software funcionem juntos corretamente. Exemplos: Sistemas Operacionais como Linux e outros.
2. Aplicativos: Programas que as pessoas usam para realizar determinado trabalho. Exemplos: Processadores de Textos, Jogos, Planilhas Eletrônicas entre outros.
3. Compiladores: O computador entende apenas uma linguagem: linguagem de máquina. Linguagem de máquina está na forma de zeros e uns. Já que é totalmente impraticável para as pessoas criarem programas usando zeros e uns, é preciso haver uma maneira de traduzir ou converter a linguagem que entendemos em linguagem de máquina, para isto, existem os compiladores.

## 1.2. Principais elementos de Hardware

---

Começaremos a falar um pouco sobre o principal para o computador funcionar, ou seja, a parte física dele.

Sem estes elementos físicos o computador não poderia funcionar. Elas interligadas formam o computador.

### 1.2.1. Memória Principal

---

A memória, onde se encontram os dados e as instruções que a CPU precisa para realizar suas tarefas, dividida em diversos locais de armazenamento que possuem seus respectivos endereços lógicos. A CPU acessa a memória pelo uso destes endereços.

### 1.2.2. Processador ou Unidade Central de Processamento(CPU)

A unidade central de processamento(CPU - Central Processing Unit), que comumente é chamada de Processador, é o “cérebro” do computador. Ele possui milhões de partes elétricas muito pequenas. Ele faz as operações fundamentais dentro do sistema. Alguns exemplos de processadores modernos são Pentium 4, Core 2 Duo, Athlon, entre outros.

### 1.2.3. Unidades de entrada e saída

Os dispositivos de entrada e saída permitem que o computador interaja com o mundo exterior pela movimentação de dados para dentro e para fora do sistema.

Exemplos de dispositivos de entradas são teclados, mouses, microfones, etc. Exemplos de dispositivos de saída são monitores, impressoras, alto-falantes, etc.

## 1.3. Lógica de programação

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o desenvolvimento.

Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

### 1.3.1. Sequência lógica ( Passos lógicos )

Agora, veremos o que seria a sequência lógica.

Os pensamentos encadeados para atingir determinado objetivo podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Podemos então falar que sequência lógica são passos executados até atingir um objetivo ou solução de um problema.

### 1.3.2. O que são instruções?

Na linguagem comum, entende-se por instruções “um conjunto de regras ou normas definidas para a realização ou emprego de algo”.

Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar.

Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica.

Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc...

É evidente que essas instruções tem que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las.

Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

Instruções são um conjunto de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.

## 1.4. Definição de algoritmo

---

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos podemos citar os algoritmos para a realização de operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo-a-passo como, por exemplo, gravar um evento.

Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas.

Em outras palavras, podemos falar também que é um processo de cálculo matemático ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições. Podemos dizer também, que são regras formais para obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas.

### 1.4.1. Exemplo de um algoritmo (Sequência lógica do dia-a-dia)

---

Os algoritmos estão presentes no nosso dia-a-dia em coisas simples, como por exemplo, ao escrever, ou abrir uma porta.

Temos como exemplo de um algoritmo:

1. “Abrir uma porta”.

- Aproximar da porta;
- Abaixar a maçaneta;
- Puxar a maçaneta com ela abaixada.

2. “Somar dois números quaisquer”.

- Escreva o primeiro número no primeiro retângulo;
- Escreva o segundo número no segundo retângulo;
- Some o primeiro número com o segundo número e coloque o resultado no terceiro retângulo.

Observe que cada um dos casos, temos 3 ações, que devem ser seguidas passo-a-passo mesmo, pois o não seguimento de uma delas, causará um erro.

Por exemplo, imagine que é construído um braço mecânico de um robô para que toda vez que alguém de aproxime, ele mesmo abra a porta. Se por acaso o passo 2 não seja colocado no algoritmo, no mínimo o nosso braço mecânico não conseguirá abri a porta, ou no pior dos casos, ele colocará força que puxará a maçaneta e a quebrará, ou a própria porta.

Por isso notamos uma grande importância os algoritmos e que eles sejam declarados cuidadosamente passo a passo.

É claro que não vamos começar construindo um braço mecânico(nem o vamos fazer) ou outra coisa mais complicada, mas antes de aprender a real utilidade dos algoritmos, iremos construir mais alguns algoritmos do dia-a-dia.



### ***1.4.2. Fazendo algoritmos - Praticando algoritmos do dia-a-dia***

---

Visto o tópico acima vamos tentar resolver e debater alguns tópicos em sala de aula com alguns tópicos do dia-a-dia:

1. Crie uma sequência lógica para tomar banho.
2. Faça um algoritmo para realizar a média entre dois números.
3. Crie uma sequência lógica para trocar um pneu de um carro.
4. Crie uma sequência lógica para fazer um sanduíche de queijo.
5. Faça um algoritmo para trocar uma lâmpada.

### ***1.5. Pseudocódigo (Forma genérica de escrever um algoritmo , utilizando uma linguagem simples)***

---

Os algoritmos são descritos em uma linguagem chamada pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo Python, estaremos gerando código em Python. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

### ***1.6. Regras para construção do algoritmo***

---

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase;
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática;
- Usar frases curtas e simples;
- Ser objetivo;
- Procurar usar palavras que não tenham sentido dúbio.

#### ***1.6.1. Estrutura principal de um algoritmo***

---

```
início
  <declarações de variáveis>
  <comandos>
fim
```

### 1.6.2. Fases

---

Anteriormente vimos que ALGORITMO é uma sequência lógica de instruções que podem ser executadas.

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

- Como fazer sanduíche, ou então, como calcular a soma de dois números.

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais:

1. ENTRADA: São os dados de entrada do algoritmo;
2. PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado final;
3. SAÍDA: São os dados já processados

### 1.7. Tipos de Dados

---

Os principais tipos de dados são: inteiro, real, caractere, lógico.

- Os do tipo inteiro é qualquer número inteiro, negativo, nulo ou positivo (-125, 0, 5, 3456);
- Os do tipo real é qualquer número real, negativo, nulo ou positivo (0.27, -0.01);
- Os do tipo lógico são conjunto de valores Falso ou verdadeiro (FALSO, VERDADEIRO);
- Os do tipo Caractere é qualquer conjunto de caracteres alfanuméricos e símbolos colocados entre aspas duplas ("Instituto Federal", "E-jovem", "7", "FALSO").

### 1.8. Identificadores

---

Os identificadores são os elementos básicos de uma linguagem onde tem como função identificar de forma única variáveis, funções, constantes entre outros

#### 1.8.1. Formação de identificadores

---

As regras para a formação dos identificadores são:

1. Os caracteres que você pode utilizar são: os números, as letras maiúsculas e minúsculas e o *underline*;
2. O primeiro caractere deve ser sempre uma letra;
3. Não são permitidos espaços em branco e caracteres especiais (@, \$, +, &, %, !);
4. Não podemos usar palavras reservadas nos identificadores, ou seja, palavras que pertençam a uma linguagem de programação.

Exemplo de identificadores válidos:

A	a	nota
NOTA	a32	NoTa1
MATRICULA	nota_1	IDADE_FILHO

Exemplo de identificadores inválidos:

5b	por começar com número
e 12	por conter espaço em branco
x - y	por conter espaço em branco e caractere especial
prova 2n	por conter espaço em branco
nota(2)	por conter caracteres especiais ()
para	por ser palavra reservada
se	por ser palavra reservada
algoritmo	por ser palavra reservada

## 1.9. Literais (ou constantes)

---

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica e literal.

## 1.10. Variáveis

---

Variável é a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a um local, ou seja, a uma posição de memória, onde pode-se armazenar qualquer valor do conjunto de valores possíveis do tipo básico associado, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante. O nome da variável deve ser um identificador válido.

### 1.10.1. Declaração de variáveis

---

As variáveis só podem armazenar valores de um mesmo tipo, de maneira que também são classificadas como sendo numéricas, lógicas e literais. Corresponde a reserva de locais na memória rotulada com o nome da variável (identificador) e cujo conteúdo será interpretado conforme o tipo declarado.

Exemplo de declaração de variáveis:

inteiro	idade, dias
real	altura
caracter	nome
lógico	filho_unico

### 1.11. Atribuição de valores

---

Para atribuir valores a uma variável, usaremos o operador de atribuição “←” (lê-se “recebe”), que tem um caráter imperativo. Exemplos de atribuição:

- idade ← 19;
- altura ← 1.80;
- nome ← “Fulano”

Erros comuns nas atribuições:

- Incompatibilidades de tipos → Tentar atribuir um tipo de dado a uma variável de outro tipo;
- Perda de Precisão → atribuir valores reais a variáveis do tipo inteiro, perdendo a parte decimal do número;
- Atribuição de valor as variáveis não declaradas.

### 1.12. Comando de Entrada e saída

---

Comandos de Entrada e saída são instruções que nos permitem, ou receber dados (Entrada) ou informar dados (Saída). O padrão de Entrada para os computadores básicos é o teclado, é dele que nós informamos o que o programa nos solicita. Já o padrão de Saída é o monitor, todas as informações mostradas ao usuário serão passadas por este dispositivo de saída. No pseudocódigo nós reservamos a palavra ler para solicitar dados, e a palavra escrever para enviar uma mensagem ou dado para o monitor, são exemplos de uso correto:

```
escrever "Digite um número: "; #Mostra no monitor a mensagem entre aspas
ler numero; #Espera o usuário digitar um número
```

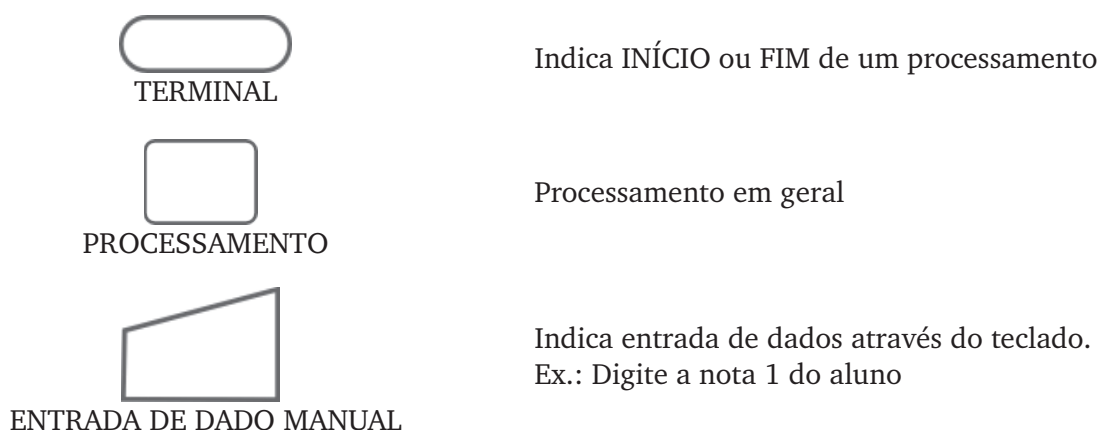
### 1.13. Diagrama de Bloco

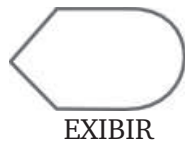
---

Uma outra maneira de mostrar um bloco de códigos seria por um diagrama básico, é muito utilizado pois é uma forma simples e eficaz para demonstrar quais passos lógicos devem ser seguidos pelo programador para chegar a um resultado.

Com o diagrama podemos definir uma sequencia de símbolos, com significado bem definido, portanto, sua principal função é a de facilitar a visualização dos passos de um processamento.

Existem diversos símbolos de um diagrama de bloco. A seguir veremos alguns símbolos básicos.

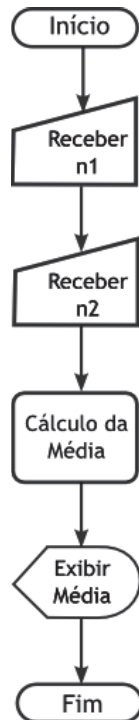




Mostra Informações ou resultados.  
Ex.: Mostrando o resultado do cálculo:

No interior do símbolo, escrevemos algum tipo de informação, caso contrário os símbolos não nos dizem nada, veja o exemplo a seguir:

Calculando a média de dois números:



Perceba que se seguirmos o fluxo de setas, podemos sem problema algum, ter o resultado, que no caso, é a média das notas.

### 1.14. Operadores Relacionais

Os operadores relacionais são utilizados para comparar *String* de caracteres e números. Os valores a serem comparados podem ser caracteres ou variáveis.

Estes operadores sempre retornam valores lógicos (verdadeiro ou falso/ True ou False) Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses. Os operadores relacionais são:

>	Verifica se um tipo de dado é maior que outro tipo
<	Verifica se um tipo de dado é menor que outro tipo
==	Verifica se um tipo de dado é igual a um outro tipo
>=	Verifica se um tipo de dado é maior ou igual a um outro tipo
<=	Verifica se um tipo de dado é menor ou igual a um outro tipo
! = ou < >	Verifica se um tipo de dado é diferente de um outro tipo

Como exemplo temos:

```
a ← 10;
b ← 20;
a > b           # retorna falso
a < b           # retorna verdadeiro
a == b         # retorna falso
```

Entre outros testes que o programador pode realizar.

### 1.15. Operadores Lógicos

Os operadores lógicos servem para combinar resultados e testes de expressões, retornando um valor cujo o resultado final é verdadeiro ou falso. Eles são:

and	Chamado também de operador “e”. Verifica se ambos os operandos são verdadeiros. Se sim, ele retorna verdadeiro, senão, retorna um valor booleano falso.
or	Chamado também de operador “ou”. Verifica se um dos valores testados é verdadeiro. Se sim, ele retorna verdadeiro, senão, retorna um valor booleano falso.
not	Chamado também de operador “não”. É usado para negar o valor de uma variável ou expressão

Suponha termos três variáveis:

```
a ← 5;
b ← 8;
c ← 1;
```

Podemos realizar as seguintes comparações e temos como resultado um valor verdadeiro ou falso.

Expressões			Resultados
a = b	and	c < a	FALSO
a != b	and	b > c	VERDADEIRO
	not	a > c	FALSO

**Exercício:** Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são verdadeiras ou falsas.

- (a) (A = C) or (B > C) ( )
- (b) not(B = A) and ((A + 1) = C) ( )
- (c) (C = (B - A)) or B ( )

## 1.16. Estruturas de Decisão e Repetição

### 1.16.1. Estrutura SE...FAÇA ISSO / SENÃO

Com o conhecimento do tópico acima, nós podemos agora, comentar sobre instruções que realizam testes entre caracteres e variáveis e também sobre laços. No nosso pseudocódigo a palavra reservada para uma instrução que realiza um teste é o “se”, seu uso é:

```
se <teste> faça isso:
    <bloco de comando>
senão:
    <bloco de comando>
```

NOTA: Perceba que a instrução “senão”, não é obrigatória, pois em alguns caso você pode necessitar fazer somente um teste que não tenha nenhum código a ser executado se o teste for falso.

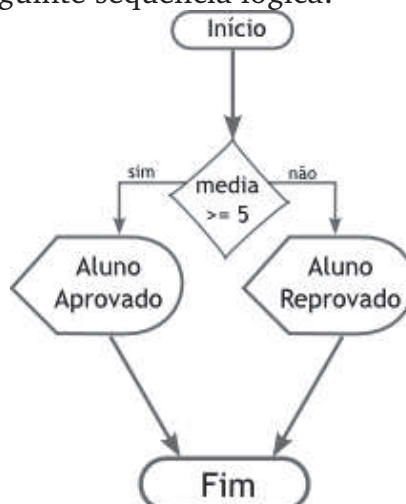
O símbolo gráfico que representa uma estrutura de decisão simples é mostrado na figura a seguir. Por um lado, o algoritmo segue se a condição testada dentro do símbolo for verdadeira, se for falsa, o algoritmo seguirá por outro caminho, o qual contém um diferente bloco de comando.



A estrutura de decisão “se” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando “se” então execute determinado comando, se tal condição não for satisfeita, execute outro determinado comando. Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5,0, veja no exemplo de algoritmo como ficaria.

```
se media >= 5.0 faça isso:
    escrever "Aluno Aprovado"
senão:
    escrever "Aluno Reprovado"
```

No Fluxograma nós temos a seguinte sequência lógica:



Exercício: Elabore um diagrama que satisfaça os seguintes pontos:

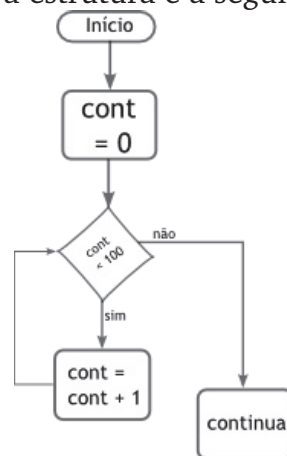
- Leia 4 (quatro) números;
- Some o primeiro com o segundo número;
- Some o terceiro com o quarto número;
- Se a primeira soma for maior que a segunda soma escreva o resultado e finalize o programa;
- Senão, imprima a segunda soma e finalize o programa.

### 1.16.2. Estrutura Enquanto

Utilizamos os comandos de repetição - neste caso a instrução “enquanto” - quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado. Em pseudocódigo sua sintaxe é:

```
enquanto <condição> faça:
    <bloco de comandos>;
fim_enquanto;
```

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores. Em diagrama de bloco a estrutura é a seguinte:



### 1.16.3. Estrutura PARA <VAR> DE <início> ATÉ <final> FAÇA

Os laços contados são úteis quando se conhece previamente o número de vezes que se deseja executar um determinado conjunto de comandos. Então, este tipo de laço nada mais é que uma estrutura dotada de mecanismos para contar o número de vezes que o corpo do laço (ou seja, o bloco de comandos) é executado. A sintaxe usada em pseudocódigo para laços contados é mostrada a seguir:

```
para <var> de <início> até <final> faça:
    <bloco de comandos>
fim_para
```

Supomos que nosso PARA, irá fazer um incremento na <var> de 1 uma unidade até que seja atingido o valor <final>.



Em termos de fluxograma, já várias representações possíveis e, até o momento não há consenso quanto à forma mais conveniente.



Algumas observações interessantes a serem comentadas:

- <var> é necessariamente uma variável, uma vez que seu valor é alterado a cada iteração (volta do laço);
- <início> e <fim> podem ser consideradas constantes ou variáveis. No segundo caso (variáveis), algumas linguagens de programação proíbem que seus valores sejam modificados durante a execução do laço;

## Capítulo 2. Scratch

---

### *2.1. Introdução ao Scratch*

---

Scratch é uma nova linguagem de programação desenvolvida pelo Lifelong Kindergarten Group no MIT Media Lab (<http://llk.media.mit.edu>), com o apoio financeiro da National Science Foundation, Microsoft, Intel Foundation, Nokia, e consórcios de investigação do MIT Media Lab.

O Scratch permite criar as tuas próprias histórias interativas, animações, jogos, música e arte - e compartilhar-las através de websites.

A programação é efetuada através da criação de sequências de comandos simples, que correspondem a blocos de várias categorias, encaixados e encadeados de forma a produzirem as ações desejadas.

Os projetos Scratch são baseados em objetos gráficos chamados **Sprites**. Pode-se mudar a aparência de um Sprite, dando-lhe um novo traje, ou fazê-lo parecer-se com uma pessoa, um objeto ou até mesmo um animal. Ainda, Pode usar qualquer imagem como traje: podes desenhar no "Editor de Pintura", importar uma imagem do disco rígido, ou arrastar uma imagem a partir de um site.

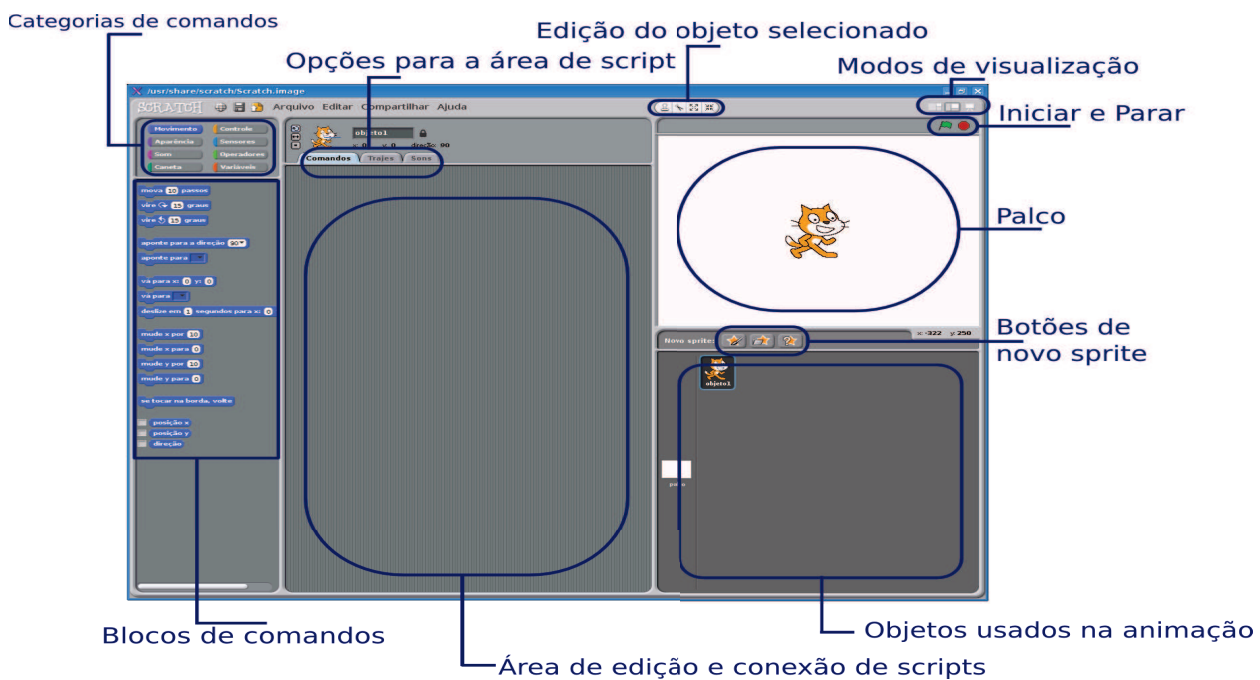
Pode-se dar instruções a um Sprite, indicando-lhe para se mover, reproduzir música ou reagir a outros Sprites. Para isso basta criar sequências de comandos, arrastando-os e encaixando-os em blocos, tal como se fossem peças de Lego ou de um puzzle. Quando clicar duas vezes (duplo-clique), o Scratch executa os blocos a partir do topo do script para o final.

Em resumo, o Scratch é um software que permite criar histórias, jogos e animações. Isto tudo de forma simples, animando o movimento de objetos.

### *2.2. Ambiente ou Interface de programação do Scratch*

---

O ambiente de programação do Scratch, é um ambiente que permite a montagem de um algoritmo de programação através de diagrama de blocos, onde, diferentemente das outras linguagens que se ensina lógica, onde tem que ser digitado comandos, o usuário do scratch tem como ver a sequência de ações tendo muito mais controle sobre o que poderá fazer.



Quando abrimos o Scratch, entramos diretamente para o modo de Edição, que nos permite criar ou modificar projetos. Veremos agora um pouco sobre a tela do Scratch e os seus principais Funções. Observe a figura a seguir a área de trabalho do Scratch. **Categoria de comandos** → Os comandos são divididos em categorias e são elas: Movimento, aparência, som, caneta, controle, sensores, operadores, variáveis;

**Blocos de comandos** → Onde estão agrupados os comandos de uma determinada categoria;

**Opções para área de Scripts** → Opções principalmente para área de Scripts, trajes e sons;

**Área de edição e conexão de Scripts (Área de Scripts ou Área de comandos)** → Edição do objeto selecionado: Nesta área é feita a programação e a união dos blocos;

**Palco** → onde os objetos são colocados e onde é possível ver o resultado da programação criada. O objeto inicial que aparece no palco é o gato;

**Modos de visualização** → São os tipos de visualização do palco;

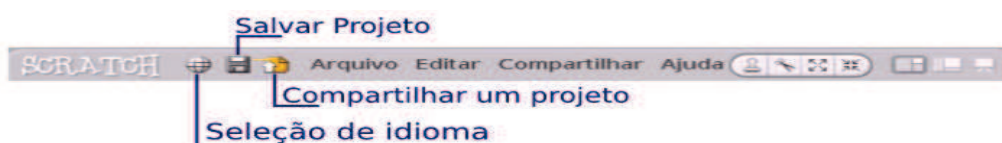
**Iniciar e Parar** → São os botões para iniciar e parar os Scripts;

**Botões de novo Sprite** → Cria um novo personagem ou objeto para o projeto;

**Objetos usados na animação (Lista de Sprites)** → Aparecem miniaturas de todos os Sprites. Clique para escolhes e editar um Sprite. O objeto em edição fica selecionado.

### 2.2.1. Menu Geral

Veremos alguns detalhes do menu geral como mostrado na tela abaixo:



**Seleção de idioma** → alterar para o idioma de preferência do usuário;

**Salvar projeto** → salvando o projeto de forma rápida;

**Compartilhar projeto** → compartilhamento do projeto para a comunidade de forma rápida, colocando-o online.

Dentro do menu geral temos os seguintes menus.

### Menu Arquivo:

Novo → Abre um novo projeto;

Abrir → Abre um projeto;

Salvar → Salvar o Projeto

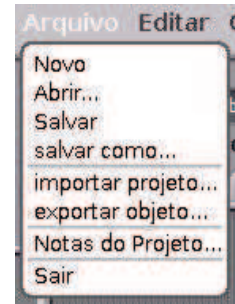
Salvar como → Salvar o projeto com outro nome

Importar projeto → Importa um projeto já existente para um projeto atual;

Exportar projeto → Exporta o projeto atual;

Notas do projeto → Insere notas sobre o projeto;

Sair → Fecha a janela do Scratch;



### Menu Editar:

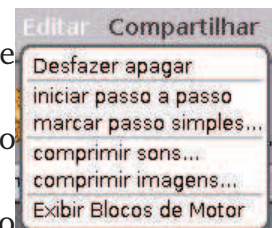
Desfazer apagar → Permite recuperar o último comando, bloco ou Sprite eliminado.

Iniciar passo a passo → Mostra o passo a passo da execução do programa.

Marcar passo simples... → Faz com que o passo a passo possa ser feito mais rápido ou mais lento.

Comprimir sons → comprime os sons usados no projeto, para reduzir o tamanho do projeto.

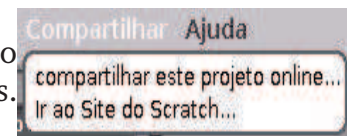
Exibir Blocos de Motor:



### Menu Compartilhar:

Compartilhar o arquivo online → permite fazer o upload do projeto para o site, para ficar disponível a todos os seus visitantes.

Ir ao site do Scratch → Vai até a página do Scratch



## 2.2.2. Alguns Elementos do Scratch

Falaremos agora mais detalhado sobre alguns dos elementos importantes dentro da janela:

### 2.2.2.1. Área de comando ou área de Script

A área de comando será onde montaremos os nossos scripts que darão vida a nossa programação que pode se tornar por exemplo um jogo. Ao lado temos um exemplo de um grupo de comandos que foram agrupadas e montado dentro da área de comandos. Mais adiante aprenderemos como montar um bloco como este.

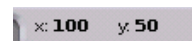


### 2.2.2.2. Palco

O palco é onde ficará e mostrará as histórias, jogos e animações. Os Sprites podem mover-se e interagir uns com os outros no palco.

O palco tem 480 unidades de largura e 360 unidades de altura e está dividido entre as coordenadas X e Y . O centro do palco tem as coordenadas 0 em "X" e 0 em "Y" .

Para saber a posição de um determinado ponto dentro do palco, mova o mouse até o ponto desejado e observe logo abaixo do palco no canto inferior direito. Pela imagem abaixo, indica-se que o cursor está no ponto X:100 e Y: 50.

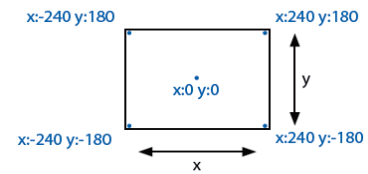


### 2.2.2.3. Mostrando sistema de coordenadas e orientações

Como falado acima, o palco tem 480 unidades de largura, e 360 unidades de altura e é dado pelo sistema ortogonal XY.

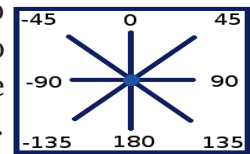
Podemos perceber isto melhor através da figura ao lado que mostra bem melhor este sistema de coordenadas.

Este sistema, o centro é dado a posição X:0 Y:0. Se percebermos bem na figura, notamos que, para cima, o Y ficará positivo e para baixo o Y ficará negativo.



A mesma coisa ocorre semelhantemente para a coordenada Y. Para a esquerda, ficará o X negativo e para direita ficará positivo.

Para sistema de direções, como mostra a figura ao lado, mostra o movimento em graus. É através das direções que vamos girar o nosso objeto dentro do palco. Guarde bem estas duas últimas figuras que mostram tanto o sistema de coordenadas como o sistema de direções. Iremos perceber para o que elas realmente servem na prática logo-logo.



### 2.2.2.4. Sprite

O Sprite é qualquer objeto do Scratch. Observe que o nosso Sprite é representado por um gato, portanto, toda vez que abrirmos o Scratch, aparecerá esta figura. Podemos ter vários Sprites dentro do palco, onde eles podem aparecer, desaparecer e fazer diversas ações onde envolvem outros Sprites para formar uma animação ou um jogo. Mais afrente aprenderemos melhor de como trabalhar com os objetos, podendo aumenta-los ou diminui-los, duplica-los ou apaga-los, ou até outras ações importantes.



### 2.2.2.5. Opções de rotação

As opções de rotação determinam como o Traje vai se comportar. Por exemplo se ele pode girar a medida que o Traje muda de direção, ou ser é só para a direita ou esquerda ou se ele não vai ter rotação alguma (o Traje nunca roda, mesmo que o Sprite mude de direção)



### 2.2.2.6. Informações do objeto

Informa principalmente sobre o nome do objeto, a posição dele em X e em Y, assim como a direção.

### 2.2.2.7. Opções para área de script

É composto de três abas que serão estudadas posteriormente:

Estas abas são: Comando (onde ficarão os comando para o nosso Sprite), Trajes (Onde ficarão os trajes e onde podemos acrescentar ou tira-los) e Sons (onde ficarão os sons que podemos usar).

### 2.2.2.8. Encolher / Crescer / Apagar / Duplicar Objeto

Como mostra na figura ao lado podemos crescer ou encolher um objeto assim como apaga-lo ou duplica-lo. Clique em um destas opções e em seguida clique no sprite e veja o que acontece.



### 2.2.2.9. Modos de Apresentação

Define como o palco será apresentado, assim ele poderá ser apresentado em pequeno tamanho, com o palco cheio ou em modo apresentação onde ele terá em toda sua tela a apresentação do palco.



## 2.2.3. Categoria de comandos

Como foi dito anteriormente, temos um grupo de 8 categorias de comandos e elas são:

**Movimento** → Determina o movimento o Sprite, fazendo ele se movimentar para área.

**Aparência** → Serve para principalmente substituição dos trajes, fazer aparecer ou desaparecer ou ainda fazer que que apareça diálogos.

**Som** → Tem como principal finalidade importar sons ou músicas.

**Caneta** → Responsável pelos traços deixados pelo objeto que está se movimentando, podendo ser modificado cor, espessura e tonalidade.

**Controle** → Possui comandos pré-definidos, responsáveis pela estrutura lógica de conexão entre outros comandos.




**Sensores** → Serve para perceber cores, distâncias e são normalmente combinados com outros comandos.

**Operadores** → Serve para fazer operações matemáticas entre outros.

**Variáveis** → Serve para criar variáveis para armazenar um determinado valor para ser usado posteriormente assim como também a questão de criação de listas.

## 2.3. Tipos de comandos do Scratch

Dentro das oito categorias disponíveis na Paleta de Comandos, há três tipos principais de comandos:

**Comandos Simples** → Estes comandos têm encaixes no topo e no fundo, como . Alguns destes comandos têm áreas onde se pode escrever números (como por exemplo o "10" do comando ) ou escolher um item a partir do menu pull-down (tal como "miau" no bloco). Alguns destes comandos têm um formato em forma de "C", onde podes inserir outros comandos, como por exemplo .

**Cabeça** → Estes comandos têm um topo arredondado, p.ex. .

Destinam-se a ser colocados no topo de blocos de comandos aguardando por ocorrências tal como "clique na bandeira verde", para que assim sejam executados os respectivos blocos de comandos.

**Valores** → Estes comandos, tais como **posição x** e **mouse pressionado?** destinam-se a serem encaixados em certas áreas dos comandos simples. Valores de formato arredondado (como **posição x** ou ) indicam números ou listas, e podem ser usados em comandos com áreas arredondadas ou retangulares (como ou ).

Os valores com extremidades pontiagudas (como **mouse pressionado?** ) indicam valores *booleanos* (verdadeiro ou falso) e podem ser inseridos em comandos com espaços pontiagudos ou retangulares (como **espere até** e **diga Olá!** ).

Alguns valores têm uma check-box junto aos mesmos. Assinalando a check-box, o valor fica visível no Palco, sendo atualizado dinamicamente à medida que for mudando. Os valores podem ser mostrados em diversos formatos:

- um pequeno mostrador com o nome do indicador;
- um mostrador largo, sem qualquer nome;
- com um slider que permite alterar o valor (apenas disponível para variáveis).

## 2.4. Mudando o Sprite(Lista de Sprites)

Quando o Scratch é aberto, no palco já está aparecendo o gato. Mas nem sempre se deseja usa-lo e então é possível inserir ou criar um novo objeto. Da mesma forma, é possível ter vários objetos em uma programação. Veja ao lado como aparece um novo objeto no palco:

Para criar ou inserir um novo objeto você deve clicar em uma das seguintes opções:

**Pintar um novo objeto** → Abre um editor que permite pintar e desenhar um objeto.

**Escolha um Sprite do arquivo** → Permite inserir um arquivo de imagem do computador.

**Pegar objeto surpresa** → Clicando neste botão, surge um objeto surpresa no palco, ou seja, a pessoa não determina o objeto que surgirá.

Os Sprites ficarão na lista de Sprites, onde podemos visualiza-los e modificar conforme desejarmos.

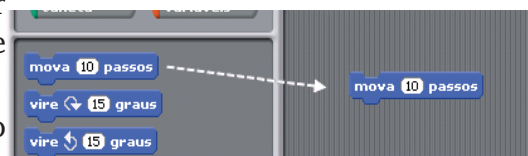


## 2.5. Movendo um comando e executando-o

De agora em diante vamos começar a estudar primeiramente os comandos básicos do scratch e depois passar a conhecer melhor suas funções.

Primeiramente devemos (como mostra a figura ao lado) mover, ou arrastar um comando para a área de scripts. Para isso é só clicar em cima do objeto desejado, segurar e movê-lo para a área de Script.

Quando abrimos o Scratch ele abre diretamente na ABA Movimento. Observe a figura acima e tente fazer o mesmo! Clique no comando “mova 10 passos” e solte na área de script.



Para executar este simples comando, clique sobre ele. Observará que após o clique no comando “mova 10 passos”, nosso Sprite se moverá. Observe que as coordenadas mudarão também.

Observe que alguns comandos, como os do exemplo acima tem campos de edição, por exemplo e para alterar o valor, clica-se na no espaço da área branca e escreve o novo número. Portanto pratique clicando no numero “10”, e altere para “-10”.

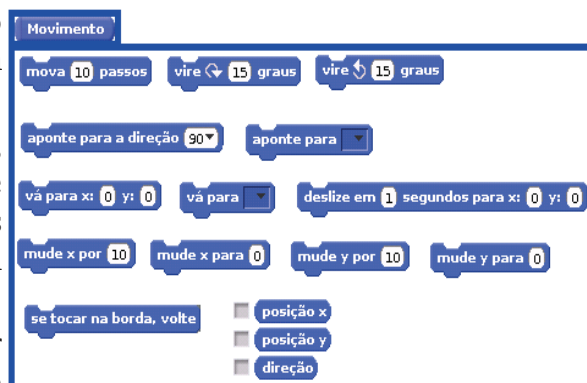
Pressione novamente o comando e perceberá que nosso Sprite e as coordenadas voltarão para o lugar de origem.

## 2.6. Trabalhando com a aba Movimento

Já que sabemos agora como mover um comando para a área de scripts e sabemos como executa-lo vamos agora aprender sobre cada uma das abas, e seus respectivos comandos.

Conforme nossa progressão no conteúdo, faremos exercícios que pegará elementos de outras abas para aprimorar mais nossos conhecimentos. Começaremos pela aba Movimento.

Esta aba é realmente onde podemos fazer nossos Sprites andarem, ou seja se movimentarem pelo palco.



### 2.6.1. Trabalhando com o mova

Começaremos a usar o comando mais comum e primordial para o Scratch, ou seja o “mova”. O comando mova, como visto acima, é o comando que dá o movimento para frente ou para trás do nosso Sprite. Se for para frente, colocar o passo positivo, por exemplo “mova 10 passos”, caso para trás, ou seja, de “costas” no caso do nosso Sprite padrão, coloca-se “mova -10 passos”. Para testar, altere o número de passos do comando que está na área de Script e observe o que acontece na área do sistema de coordenadas. Para altera-lo, clique uma vez em cima do numeral “10” e substitua por “-10”.

### 2.6.2. Trabalhando com o vire

Percebe-se que com o mova, o nosso objeto somente se movimenta no eixo X (ou seja, para frente e para trás). Precisamos de alguma ferramenta que faça com que ele faça o movimento entre o eixo Y também. Para isso iremos utilizar a rotação através do comando “Vire.”

Lembre-se que temos um sistema de coordenadas e de direção. Visto estes 2 sistemas anteriormente, podemos dizer que o comando mova está diretamente ligado ao sistema de coordenadas e o comando vire está diretamente ligado ao sistema de direção, ou seja, com o vire trabalhamos o giro. Este giro será dado em graus.

Para testar, mova os dois comandos de vire (Figura ao lado) para a área de Script. Faça o teste clicando algumas vezes no primeiro vire e observe que nosso Sprite vai girando. Observe também em quantos graus está o objeto nas informações do Sprite.





Atenção: Volte para o tópico “Mostrando sistema de coordenadas e orientações” e perceba que o sistema de graus, não corresponde ao que normalmente usamos. Tente girar ora para um lado, hora para o outro e veja por fim o que acontece. Após fazer os testes, deixe o Sprite na direção 90.

Dica: Podemos testar o uso de um comando sem colocar na área de script. Para isso, somente clique em cima em um dos dois comando( Pode clicar primeiramente do de cima e depois no de baixo ) . Observe o que acontece e depois volte para a direção “90”.

### 2.6.3. Alterando o numero de graus e passos no comando

Como já foi cimentado no tópico 2.4.1, podemos alterar ou o numero de passos ou o numero de graus.



A primeira alternativa é modificar o numero diretamente no próprio bloco de comandos. A segunda é alterar dentro da área de Script. A diferença entre a primeira e a segunda é que, se modificarmos dentro do bloco de comandos, se arrastarmos o comando para dentro da área de Script, todos eles sairão com aquele numero. Por exemplo, se trocarmos o “10” de “mova 10 passos” por “20”, qualquer comando igual a este que arrastarmos para área de Script, sairá não mais com o “10” e sim com o “20”. Diferentemente , se alterarmos um que já está dentro da área de Script, será alterado somente aquele comando. Estas alterações servirão para outros comandos que usam o mesmo conceito.

### 2.6.4. Movendo e virando

Já que temos a noção inicial do “mova n passos” e “vire m graus”, onde N e M são numerais quaisquer que vão simbolizar o número de graus e passo dados, vamos começar a dar uma movimentação maior ao nosso Sprite.

Mova diversos comando de “mova 10 passos” e “vire 15 graus”, modificando para valores diferentes. Clique neles e vá observando o comportamento do Sprite se movimentando na tela.

### 2.6.5. Mostrando como voltar para a posição inicial

Possivelmente como seu Sprite deve ter se movimentado bastante no exercício anterior, ele deve estar totalmente “desorientado”, por exemplo, imagine que nosso Sprite está com: x:191 y:-177 direção: -130 (Faça o teste para ver onde nosso Sprite vai estar). Se for observar, pode notar que nosso objeto está totalmente “perdido.”

Seria uma das maneira de arrumar melhor nosso Sprite, clicar nele e arrastar para x:0 y:0, porem iria ficar trabalhoso, ter que ficar mudando o numero de graus, e ainda sim um pouco das variáveis X e Y para chegar na direção e coordenadas exatas x:0 y:0 direção: 90.

Vamos facilitar as coisas usando mais comandos dentro da aba movimento. Observe a figura ao lado e veja estes comandos. Arraste-os para a área de Script e clique em ambos. Como percebe, é a forma fácil de voltar a posição inicial.



Veremos que daqui para frente, alguns comandos tem "pull-down", por exemplo um comando que veremos mais afrente como . Clique na seta do comando “aponte para a direção” para ver o menu. Escolha a opção desejada clicando nela.

### 2.6.6. Trabalhando mais com a posição e a orientação

---

Podemos explorar outros blocos ainda dentro de movimento, como por exemplo:

- Fazer com que o nosso Sprite deslize por uma certa quantidade de segundos pelo palco;
- Fazer com que ele mude o valor de x e/ou y andando uma certa quantidade de passos;
- Fazer com que ele mude a posição do x e/ou y pelo valor determinado;
- Fazer com que seja mostrado dentro do palco a posição de X e/ou Y e/ou direção;
- Fazer com que se o Objeto tocar na borda ele volte;

Volte ao tópico 2.4. e tente descobrir quais são os comandos que executam cada uma das ações acima. Dialogue junto com seus colegas e junto com o seu Instrutor a melhor maneira de utiliza-los

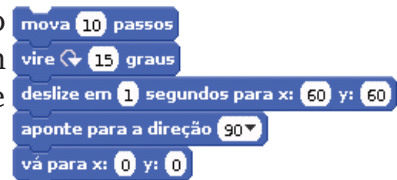
### 2.7. Trabalhando em blocos

---

Até agora, trabalhamos praticamente com comando individuais, porem, teremos que agora começar a aprender como junta-los para formar um Script. Ou seja, para ficar mais claro, um Script vai ser um conjunto de ações que vamos reunir para fazer nosso Sprite se movimentar.

Para isso vamos abrir um novo projeto sem salvar o anterior. Clique em Arquivo → Novo. Aparecerá uma janela para salvar o projeto atual, porem como não vamos utiliza-lo, clique em não e aparecerá um novo projeto.

Agora, arraste 2 blocos de ações e coloque um bem próximo ao outro. Quando soltamos, perceberá que como se fosse um “ímã” os 2 ficarão grudados. Observe a figura ao lado e monte este grupo de ações e execute. Perceba o que acontece.



### 2.8. Retirando uma ação de um bloco

---

Para a retirada de uma ação do bloco é simples, segure esta ação e puxe-a para baixo, por exemplo e perceberá que ele vai se desprender do bloco.

### 2.9. Removendo uma ação

---

Digamos que não queremos mais uma ação ou um Script (que como já sabemos é um conjunto de ações). Para remover da área de Script, simplesmente pegamos a ação ou o bloco não mais desejada e “puxamos” de volta para o bloco de ações. Com isso, perceberá que o grupo ou a ação vai desaparecer.

### 2.10. Renomeando um objeto

---

Observe dentro da área de informações de Script que temos escrito “objeto1”. Para alterar o nome, por exemplo, queremos colocar o nome do objeto como “gato”. Clique em cima e dê um duplo clique. Modifique para o nome desejado. Observe dentro da área de objetos usados que também modificou o nome.

## 2.11. Trabalhando com aba aparência

### 2.11.1. Trabalhando com Trajes

Cada Sprite pode ter diferentes “aspectos” que são como de fossem partes de um movimentos, o que chama-se de Trajes. Clicando-se na aba Trajes. Clicando no separador "Trajes" podes ver quantos aspectos o Sprite tem. Podemos através disso, Pintar um novo Traje, Importar um novo traje ou ainda usar a câmera(Webcam) para fotografar para o traje.

Caso queira-se “Mexer” no Sprite, podemos, Editar ou Copiar, ou Apagar o Sprite.

Com os trajes e os comandos do exemplo abaixo, podes criar animações.



Para a opção de traje temos os comandos “mude para o próximo traje” e “próximo traje”, além de poder mostrar no palco qual é o traje que está sendo usado (Observe-os na aba aparência e tente manipula-los com eles). Posteriormente abordaremos melhor sobre como manipular melhor com os Sprites.

### 2.11.2. Trabalhando com “Diga” e “Pense”

Podemos fazer nossos objetos “Falarem” ou “Pensarem” através dos comandos “diga” e “pense” onde pode ser falado rapidamente ou por alguns segundos.

Arrasta para a área de Blocos um dos quatro comandos da categoria "Aparência" que permitem a criação de balões com as falas. Podes alterar o que queres que o sprite diga ou pense no próprio comando. Podes definir o tempo de duração das falas, ou usar comandos sem tempo definido.

### 2.11.3. Mudando os efeitos

Podemos ter vários tipos de efeitos para alterar o nosso sprite:

**Cor** → Modifica a cor do nosso sprite

**Olho de peixe** → Modifica o formato, fazendo com que se o valor colocado for muito grande, fica com a forma de um olho de peixe

**Girar** → Deforma nosso sprite em forma de redemoinho

**Pixelar** → Faz com que nosso sprite de deforme até desaparecer dependendo do que for feito

**Mosaico** → Faz nosso objeto “se multiplicar” em pequenos tamanhos

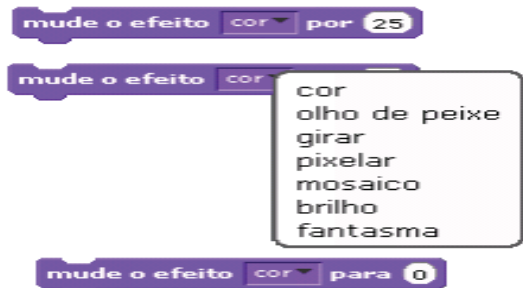
**Brilho** → Modifica o brilho do nosso sprite

**Fantasma** → Faz com que nosso sprite vá desaparecendo.

Para realizar um destas ações acima, arraste para a área de script “Mude o efeito” e escolha o que queres que aconteça, usando o menu "pull-down"(Observe a figura ao lado).

Caso esteja só testando, ou no seu projeto queira tirar(limpar) os efeitos, clique no comando

**limpe os efeitos gráficos**



Dica: Tanto nestes blocos de efeito como em outros, podemos ter blocos com “por” e outros com “para”. Por exemplo: Na aba movimento temos o bloco “mude x por” e “mude x para”. Neste primeiro caso, cada vez que clicamos no bloco, o valor de x vai sendo incrementado, dependendo do seu valor, ou seja, digamos que temos “mude x por 10”, então toda vez que clicamos, o x vai sendo incrementado em 10. Com o “mude x para” o valor de x será constante para aquele valor determinado, ou seja se colocarmos um bloco “mude x para 10”, não importa quantas vezes clicamos, pois ele sempre ficará na posição 10.

#### 2.11.4. Modificando os tamanhos

Modifica o tamanho do nosso Sprite. Estes comandos equivalem ao crescer objeto e encolher o objeto (visto anteriormente). Podemos mostrar no Palco a porcentagem que ele está, alterar para certa porcentagem o tamanho, ou que ele vá aumentando ao clicar.



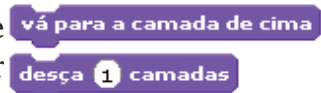
#### 2.11.5. Aparecendo / desaparecendo

Estes comandos fazem com que o Script apareça e desapareça.



#### 2.11.6. Trabalhando com camadas

Para os casos que precisa-se alterar a ordem dos sprites, por exemplo, um sobrepor o outro ou uma forma, temos a opção de fazer isso através dos dois comandos ao lado, onde podemos fazer com que o nosso objeto suba uma camada (primeira ação), ou podemos determinar o numero de camadas a serem alteradas (através da segunda ação)



### 2.12. Trabalhando com a aba de som (adicionando som)

Já aprendemos até agora como dar movimento para o nosso objeto e modificar parâmetros da aparência dele. Vamos começar a colocar sons neste nosso objeto.

Podemos primeiramente importar um arquivo já existente da própria biblioteca do Scratch ou gravar um som, podendo executar, parar ou excluir.



#### 2.12.1. Trabalhando com o comando Toque o som

Começaremos com o comando “toque o som” onde tocará o som que queremos, seja ele uma gravação feita ou um som já gravado.

Para usar o som escolhido na sua programação escolha o bloco “toque o som” e encaixe no seu script.



Atenção: se o som não funcionar, verifique se este recurso funciona no seu computador (se o som está ligado e existem caixas de som funcionando).

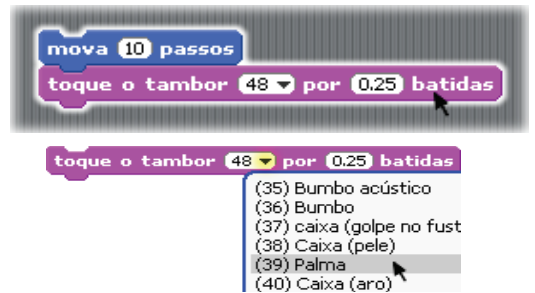
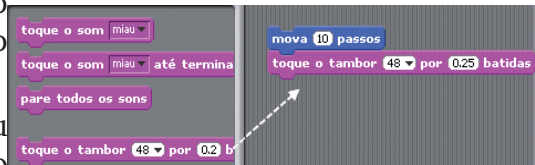
### 2.12.2. Trabalhando com Toque o tambor e Toque a nota

Para colocar som no script (um som de instrumento ou outro), você pode usar o comando “Toque o tambor”. Ele fica disponível na categoria som.

Você pode usar este bloco de comando sozinho ou agrupado com outros comandos, como mostrado anteriormente em outros scripts. Como já sabemos e só clicar e arrastar o bloco para a área de edição de Scripts. Se for o caso, encaixe este bloco com os já existentes no script.

Para ver o funcionamento, dê um duplo clique sobre o grupo de blocos.

Para escolher o som desejado, clique na seta destacada e escolha entre as opções do menu.

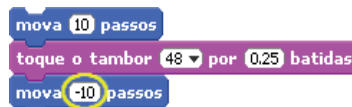


### 2.12.3. Trabalhando com volume e ritmo

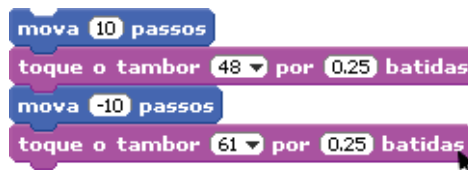
## 2.13. Ensaizando um pé de dança

Aprendemos anteriormente os recursos de movimento e toque. Usaremos estes conhecimentos para fazer um movimento que engloba movimento e som.

Primeiramente arraste 2 comandos mova e um comando toque o tambor. Altere um dos mova para o valor “-10”, como mostra a figura abaixo.



Depois também é possível acrescentar outro bloco de “toque o tambor” após o segundo movimento. Vamos então incrementar um pouco mais, e colocar um outro toque, como mostra a imagem abaixo. Dê um duplo clique sobre o Script e perceba o que acontece.



Agora, implemente o Script semelhante ao da figura ao lado, introduzindo o bloco “sempre” localizado na secção Controle.

Clique em cima e veja o que acontece.



Para interromper o programa use o botão vermelho no canto superior direito do palco, ou clique novamente sobre o script.



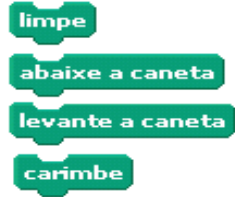
## 2.14. Trabalhando com a aba caneta

Até agora aprendemos como modificar a aparência do nosso personagem, faz-lo falar e até tocar um som. Agora é importante começar a pensar: como seria fazer com que fique registado na tela algo quando o nosso elemento se mexer? Por exemplo, podemos querer desenhar figuras geométricas, como quadrados triângulos, entre outros.

Por isso vamos aprender um pouquinho sobre a aba caneta e seus componentes.

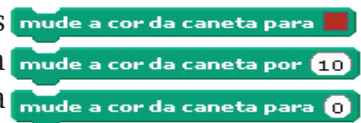
### 2.14.1. Usando o Abaixar, levantar, carimbe e limpe

Estas são as principais ações desta aba. Através delas podemos fazer com que o nosso objeto ao passar, deixe algum “rastros” da sua passagem, ou em determinado momento deixe de passar (como na construção de um caminho), usar o carimbe, que marca a passagem do elemento (é marcado no palco uma imagem do elemento) ou ainda limpar tudo, deixando a área limpa.

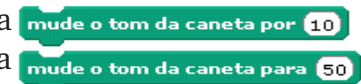


### 2.14.2. Mudando a cor, o tom e o tamanho da caneta

Podemos além de fazer nosso objeto desenhar na tela, podemos alterar os parâmetros desta linha, por exemplo, mudar a cor da caneta para uma cor específica, ou variar o valor da caneta (utilizar os 3 primeiros comando ao lados).



É possível ainda fazer a modificação do tom desta caneta (para isso, utilize os 2 blocos seguintes ao lado), onde vai variar a tonalidade da cor escolhida para mais escura ou mais clara.



Por último, mudar o tamanho do traço, podendo ser maior ou menor, dependendo do valor registrado.



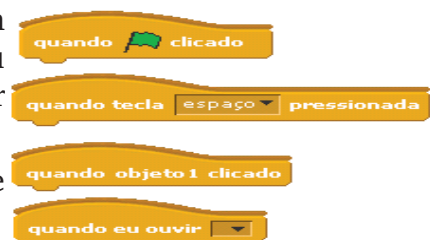
## 2.15. Trabalhando com a aba controle

Como próprio nome já sugere, a aba controle permite a quem está fazendo os scripts um controle maior sobre o que está fazendo, onde poderá fazer com que um bloco seja inicializado ao pressionar uma tecla, ou desde a inicialização de todo o programa. Até outras ações como executar várias vezes uma determinada ação. Veremos abaixo mais sobre.

### 2.15.1. Usando o quando

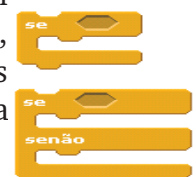
Através destes comandos, é possível normalmente iniciar um bloco quando por exemplo, o programa for iniciado, ou quando o objeto for pressionado, ou ainda quando for pressionado uma tecla (determinada por quem fez o script).

Ao lado temos os principais comandos que ficam no topo de cada script.



### 2.15.2. Usando os controles se / se senão

Através destes blocos é possível fazer o teste de condições, por exemplo, um caso bem clássico seria fazer a verificação se um objeto está tocando no outro, ou na borda (Será visto mais a frente). Portanto nestes comandos normalmente existem um conjunto de outros dentro, onde se satisfizer a condição, realizará os comandos dentro, senão, não realizará.



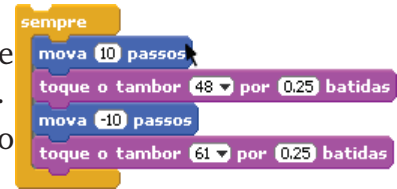
### 2.15.3. Uso do controle sempre

Como foi visto anteriormente, é possível programar no Scratch que uma ou mais ações continuem acontecendo e se repetindo por tempo indeterminado. Para isso se usa o comando SEMPRE, disponível na categoria Controle.

Clique e arraste o bloco SEMPRE para a área de edição de scripts. Encaixe o grupo de comandos dentro do bloco SEMPRE.

Para arrastar um conjunto de blocos, clique sobre o primeiro bloco (no topo do conjunto) e arraste tudo.

Para parar a programação após usar o comando SEMPRE, clique no botão vermelho que significa Parar Tudo.



### 2.15.4. Usando os controles repita / repita até / sempre se

O comando sempre se, é realizado sempre que a condição seguinte for verdade. Por exemplo, seria dizer “Sempre se tecla x for pressionada”, ou seja, só será executado o que tiver dentro se o que tiver depois for satisfeito. Neste caso, é diferente do caso acima, pois o bloco será executado sempre, mas neste caso, ele vai executar o que tem dentro do bloco cada vez que a tecla for pressionada.

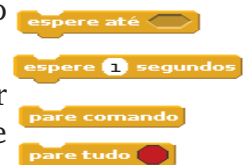
A condição repita já é um pouco diferente. O repita faz com que uma ação dentro do bloco seja realizada N vezes, ou por exemplo, seja repetida até (repita até) uma tecla for pressionada (neste caso quando a tecla for pressionada a ação irá parar)




### 2.15.5. Usando o pare e espere


Estes comandos (pare e espere) normalmente servem, como o próprio nome já diz para parar ou para que espere até uma determinada ação.

O espere pode servir para que se espere tantos segundos, ou para esperar até que uma determinada coisa seja feita (pressionado o botão do mouse ou pressionado uma tecla.)



## 2.16. Iniciando e parando um programa

Uma das maneiras de iniciar ou parar os programas é através dos botões  onde, a bandeira verde, inicia o programa e o botão vermelho para um programa.

O Scratch também possui controles para o início da execução dos scripts. Um exemplo é a bandeira verde que fica sobre a tela de visualização das programações: ela pode ser usada para iniciar o funcionamento de um script. Para isso é necessário que seja colocado no script o bloco de controle que indica .

Clique no bloco e arraste para a área de edição de scripts. Encaixe o bloco sobre o conjunto já existente, se for o caso. Este controle deve ser o primeiro em um grupo de blocos, pois ele que determina o início desta execução.



Para testar, clique sobre a bandeira verde que significa Iniciar Scripts.

A **Bandeira Verde** fornece uma maneira conveniente de iniciar vários scripts ao mesmo tempo. Clica na Bandeira Verde para iniciar todos os blocos com no topo. No "Modo de Apresentação" a Bandeira Verde, surge como um pequeno ícone no canto superior direito do ecrã. Pressionar a tecla "Enter" tem o mesmo efeito que clicar na Bandeira Verde.

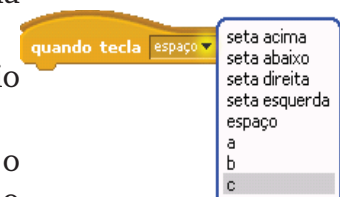
## 2.17. Usando Teclas – Detectando teclas pressionadas

Para iniciar um script, além de usar a bandeira verde é possível determinar uma tecla do teclado que funcione como disparadora do script. Desta forma, quando a tecla for pressionada o script inicia sua execução.

Para determinar que o início da execução será definido por uma tecla, você precisa colocar no início de um script o controle .

Arraste o bloco para a área de edição de script e o encaixe no início de um conjunto de blocos.

Para determinar qual tecla do teclado será usada para iniciar o script, clique na seta(ver imagem ao lado) e escolha a opção desejada.



Você pode usar um controle inicial de Script diferente para cada conjunto de blocos. É assim que se faz para determinar movimentos diferentes de um objeto de acordo com o clique nas setas de direção do teclado.

## 2.18. Mudando a cor de objeto quando pressionada determinada tecla

Iremos ver de forma prática a mudança de cor em um pequeno Script. Na figura ao lado temos 2 blocos. O primeiro muda o efeito de cor, enquanto o segundo irá retornar para a cor primária.



Monte os blocos e veja o que acontece.

Em vez de escolher a tecla "A" pode escolher outra tecla qualquer. Basta selecionar a tecla pretendida nas opções do próprio bloco.

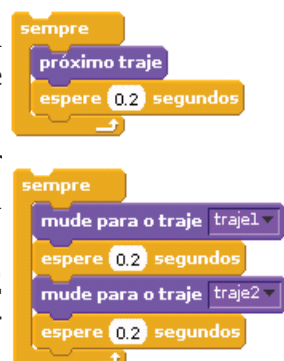


## 2.19. Mudando os trajes com controle

Para fazer uma animação, no Scratch é bastante simples. Já vimos algo sobre isso no tópico "Trabalhando com trajes", agora veremos melhor como automatizar as coisas através do controle. O efeito final é o mesmo de uma imagem gif, ou seja, imagens que se mechem, onde aparecem diferentes posições de um personagem e a troca das imagens das posições produz a ideia de animação.

Escolha o objeto que será animado e clique em *trajes*. Você pode criar as diferentes posições do objeto desenhando o novo a partir do inicial (fazer uma cópia do original e editar) ou importar as posições.

Depois faça o script do objeto que será animado. Use o bloco SEMPRE e dentro dele o bloco "próximo traje". Este bloco faz o objeto alternar entre seus trajes já criados.





É importante colocar um tempo após a troca de traje para que seja possível visualizar a troca, ou isso acontecerá muito rápido.

**Exercício:** Coloque os dois Scripts na área e tente analisa-los. Veja se tem ou não outras possibilidades de fazer isso. Altere estes Scripts para outras formas e discuta em sala de aula com os colegas e com o instrutor o que foi aprendido.

## 2.20. *Tocar na borda e voltar*

Quando você faz algumas programações no Scratch, é importante que o objeto ao tocar na borda do palco volte. Um exemplo disso pode ser uma bola que rola, bate na borda e volta.

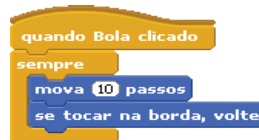
Puxe o bloco “mova” para a área de edição de Scripts.

Pegue o bloco “sempre” e coloque na área de edição de Scripts.

Encaixe o “mova” dentro do “sempre”.

Pegue o bloco “se tocar na borda, volte” na categoria Movimento e coloque dentro do SEMPRE.

Se você quiser que a bola comece a andar quando for pressionada pelo mouse (clorada), use o controle abaixo:



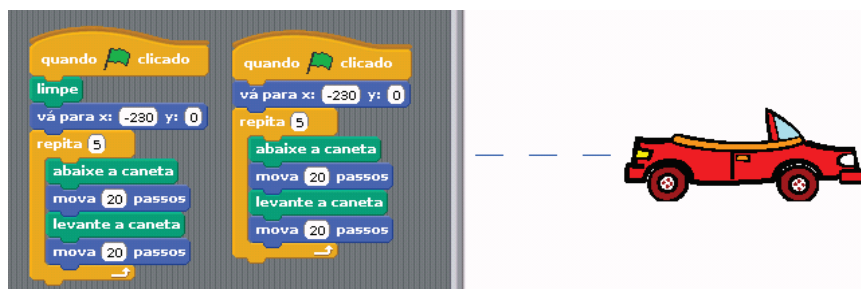
Você também pode determinar que o Script inicie quando a bandeira verde for pressionada.

## 2.21. *Desenhando uma estrada*

O Sprite, ao mover-se, pode deixar “rasto”. Para tal, antes de mandar executar os comandos de movimento, é preciso selecionar o bloco abaixe a caneta.

Para que o Sprite deixe de desenhar, use o bloco levante a caneta.

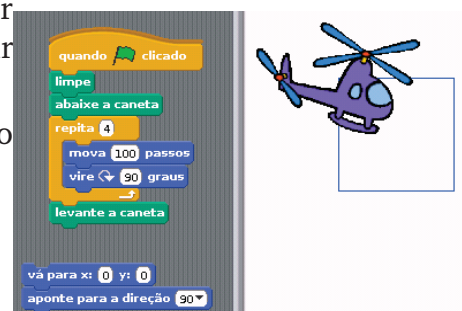
Para limpar os efeitos gráficos do palco, utilize o bloco limpe.



## 2.22. *Desenhando um quadrado*

Observe o Script ao lado e tente fazer o mesmo para criar quadrados. Altere este Script de modo a desenhar quadrados de diferentes tamanhos.

Lembre-se que para alterar a a posição e orientação do Sprite utilize os comandos indicados no bloco separado .

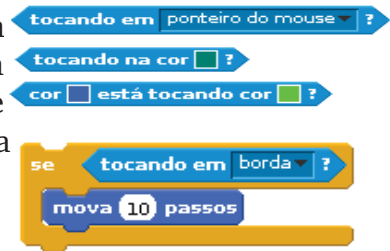


**Exercício:** A partir do Script anterior, faça as alterações que achar necessárias para desenhar um triângulo, um pentágono, um hexágono, uma circunferência.

### 2.23. Trabalhando com a aba Sensores

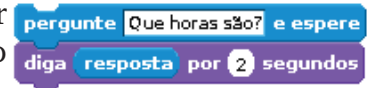
#### 2.23.1. Comandos tocando em / tocando na / tocando cor

Através destes comandos agora podemos utilizar melhor os nossos comandos de controle, por exemplo, podemos agora saber se o nosso objeto está tocando em uma determinada cor, ou no ponteiro de mouse ou na borda por exemplo, ou se uma cor está tocando em outra cor. Para testar, construa algum script usando o bloco ao lado com o controle SE.



#### 2.23.2. Comandos pergunte e resposta

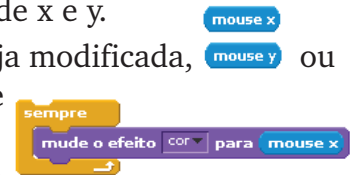
É possível também fazer perguntas e exibir as respostas por algum tempo no nosso palco. Vamos construir a estrutura ao lado e veja o resultado.



#### 2.23.3. Comandos mouse x e mouse y

Estes dois comandos tem como característica informar os valores de x e y.

Digamos que se queira ao mover o mouse a cor do nosso sprite seja modificada, ou utilizar outro efeito quando o mouse se mover pela tela. Observe o bloco ao lado e tente fazer alterações com outros efeitos.



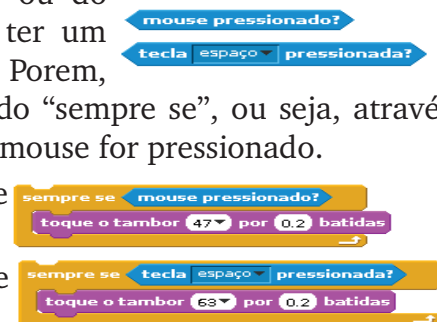
Outra maneira que podemos utilizar é praticamente com o mesmo bloco, mas ao invés de utilizar o “Mude o efeito”, utilize o “diga”

#### 2.23.4. Comandos mouse pressionado / tecla pressionada / temporizador

Estes comandos vão identificar alguma ação do teclado ou do mouse. Assim como os exemplos anteriores, temos que ter um comando de controle que normalmente é o “sempre”.

Porém, vamos modificar um pouco e fazer a utilização do comando “sempre se”, ou seja, através disso, a ação não será feita sempre, e sim só se a tecla ou o mouse for pressionado.

Observe os blocos ao lado e faça-os e tente ver o que acontece.

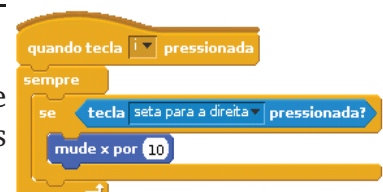


Já o temporizador funciona como um cronometro, onde desde a abertura do programa, vai “correndo” o tempo.

### 2.24. Movimentando Sprite por Sensores

Para melhor movimentar nosso Sprite, observe o ao lado

Tente fazê-lo e executar. E veja o que acontece. Observe que este script só vai em uma direção, ou seja, ele move 10 passos somente para a direita.



Com base nisso, faça o mesmo por todos os lados (para baixo, para esquerda e para cima). Coloque também uma ação para que quando pressionada a tecla “P”, pare o Script.

### Trabalhando com a aba Operadores

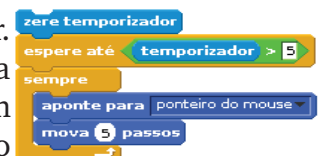
Dentro da aba de operadores, temos primeiramente os operadores de operações matemáticas (como soma, subtração), depois os operadores de comparação (maior, menor e igual), operação de palavras (mostrar o número de letras, ou para juntar palavras), arredondamento ou mostrar o resto de uma divisão entre outros. Observe as figuras de comandos abaixo e tente usar eles com os blocos estudados até agora.



### 2.25. Movendo-se com relação ao temporizador

Este exemplo dá a ideia de como vamos usar o temporizador.

Desde que o Scratch é iniciado, o temporizador começa a funcionar. Porém temos a opção “zere temporizador” se quisermos recomençar a contagem. Observe o exemplo ao lado e o implemente. Discuta com os colegas o que ele faz e tente ver outras maneiras de usar o temporizador com os conhecimentos adquiridos até aqui.

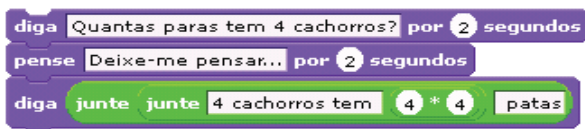


### 2.26. Fala e Calcular

Como visto anteriormente, o nosso sprite pode falar através do comando diga. Como já foi visto, nele poderá se determinar o quê será dito e o tempo que essa mensagem ficará aparecendo.

Vamos usar um simples script que faça um simples calculo já pré-determinado:

Monte o bloco abaixo para ele falar e calcular



### 2.27. Trabalhando com a aba Variáveis

A aba variáveis, vai nos proporcionar trabalhar com valores que variam com o tempo, fazendo com que um certo valor seja armazenado temporariamente para ser usado futuramente. Através da aba variável, podemos inicialmente criar uma, especificando em seguida o nome desta variável.

NO caso ao lado, denominamos nossa variáveis com o nome variáveis, onde pode-se mudar o valor da variável para um valor, ou por um valor (neste segundo caso, o que tem em variável caso seja um valor, será incrementado a cada passagem, ou clique, pelo bloco)



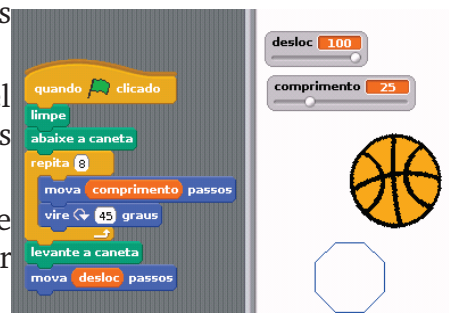
### 2.27.1. Usando variáveis

Construa um Script que desenhe quadrados cujos lados podem variar de comprimento.

Para tal, recorra à secção Variáveis e defina a variável “comprimento” que, neste caso, pode assumir os valores de 0 a 100.

Antes de mandar executar o programa, faça duplo-clique sobre a caixa correspondente à variável fazendo surgir uma menu deslizante.

Arraste a bolinha branca, selecionando um valor para o comprimento.

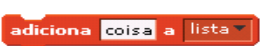


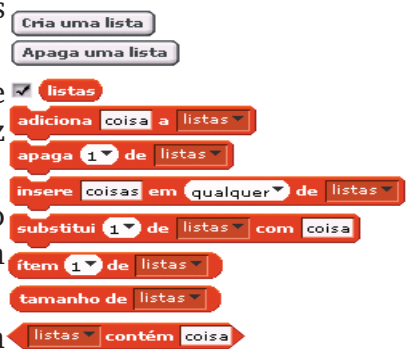
### 2.27.2. Listas

No Scratch é possível criar e manipular listas de itens compostas por números e frases.

Para criar uma lista, é preciso ir no bloco de “Variáveis” e depois em “Criar uma Lista” dando um nome a ela. Uma vez a lista criada, vários blocos irão surgir.

Quando cria-se uma lista ela fica visível no palco, mostrando todos os seus itens. É possível editar os itens da lista diretamente no palco.

A lista é criada vazia, com a indicação de "itens: 0". Para adicionar itens à lista, clica no símbolo "mais", no canto inferior esquerdo do mostrador. Como alternativa, pode-se adicionar itens usando comandos simples (exemplo:  ).



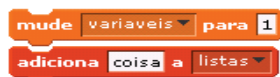
É possível ajustar o tamanho do espaço da listar, arrastando-lhe o canto inferior direito.

Pode-se ainda ter a opção de querer exportar a lista para um documento TXT clicando com o botão direito que será criado diretamente na pasta de instalação do Scratch.

Inversamente podes usar a opção disponível para importar dados para uma lista a partir de qualquer ficheiro .TXT com um item por linha.

### 2.28. Caracteres

Podem formar-se conjuntos de caracteres usando letras, algarismos ou outros caracteres.



Os conjuntos de caracteres podem ser guardados em variáveis ou listas.

Podes comparar conjuntos de caracteres através dos seguintes comandos:



A comparação é feita a começar nos primeiros caracteres. Os algarismos são os mais pequenos, seguidos dos caracteres especiais, das letras maiúsculas e finalmente das letras minúsculas, por esta ordem. No caso dos primeiros caracteres serem iguais, são comparados os segundos caracteres e assim sucessivamente; porém se um dos conjuntos apenas tiver algarismos, então será tratado como número e a comparação não é possível.

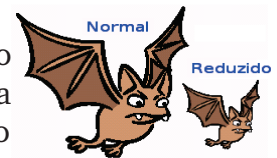
## 2.29. Criando um pequeno aplicativo

Usando todos os conhecimentos anteriores, vamos criar um mini-aplicativo que terá os seguintes elementos: uma bola, a figura de dois jogadores (Jog 1 e Jog 2) e onde cada um pode fazer “gol” no outro.

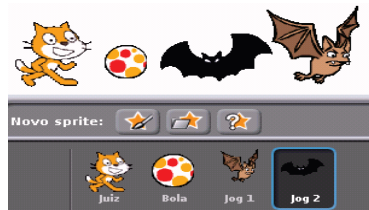
Primeiramente, vamos escolher os nossos Sprites, portanto, escolha o Sprite do arquivo “beachball1” dentro da pasta “things”. Depois, encolha o objeto para ficar em um tamanho relativamente pequeno e renomeie para o nome “Bola”.

O nosso “objeto1” será o nosso juiz, então nomeie para “Juiz”.

Escolha também os Scripts “bat1-a” e “bat-1b”, renomeando respectivamente para “Jog. 1” e “Jog. 2”. Diminua todos os objetos para ficar praticamente do mesmo tamanho da bola. Observe a figura ao lado o tamanho real e o reduzido como base.

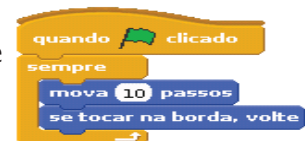


Na imagem abaixo temos a ideia de como ficará os nossos elementos.



Vamos começar a programação da nossa bola.

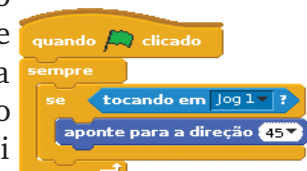
Primeiramente vamos dar o movimento a esta bola, portanto, clique na bola na área de Sprites e coloque o Script ao lado.



Para muitas programações, jogos e histórias é importante usar testes.

Podemos fazer uma bola bater em um objeto e quando ela bater, voltar. Mas como ela vai saber que bateu? Como determinar o que acontece quando ela bate?

Vamos começar pela programação do Jogador 1 (Jog 1). E para isso monte o Script ao lado. Após montado, duplique o Script e troque “Jog 1” por “Jog 2” e a direção 45 pela direção -45. Clique na Bandeira Verde e veja o que acontece. Perceba que quando a bola toca no Jogador 1, ela vai na direção 45 e quando toca no Jogador 2 ele vai na direção -45.



Podemos também fazer com que a bola só se movimente quando clicada nela, para isso, use o controle “quando Bola clicado”.

No Script perceba que como tem o “se tocar na borda, volte”, quando toca na borda (qualquer borda do palco) ela volta na direção contrária.

Veja que foram feitos três Scripts separados: um para o movimento da bola e outros para o movimento dos “jogadores”. Também é possível fazer tudo junto, usando apenas um controle de início do Script e apenas um bloco SEMPRE.

**Exercício: Modificar os Scripts, para que os três tornem-se só um.**

Agora vamos complementar o nosso projeto. O desafio é fazer uma bola ir na direção do “gol” oposto e se bater nele, dizer “Ponto para o Jogador 1” ou “Ponto para o Jogador 2”.

Para este exemplo foram importados os objetos button e buttonPressed e chamados de “Gol 1” e “Gol 2”. Rotacionamos os botões para ficar como mostra a figura ao lado.

Vamos agora fazer com que nossa bola sempre parta de um ponto específico, ou seja, que parta toda vez que iniciar do ponto x:0 y:0 apontando para a direção 0 graus. Implemente isso entre o os comandos “quando” e o “sempre”.

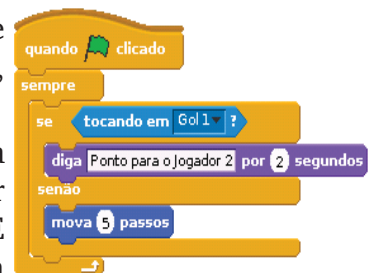
Agora, devemos fazer com que quando tocar em um dos Gols, ele diga isso.

Acrescente na área de edição de Scripts o bloco “se, senão” que fica na categoria Controle.

Coloque no espaço do “se” o sensor “tocando em ...”. e selecione o “Gol 1”.

Você pode colocar dentro do “senão” o comando MOVA para que a bola ande se não tocar no Gol 1. Mas se deixar o senão vazio, apenas não acontecerá nada quando a bola *não* tocar no Gol 1.

Dentro do bloco “se” coloque a ação que deve ocorrer quando a bola tocar o gol, ou seja, coloque o bloco “Ponto para o Jogador 2 por 2 segundos”. Coloque o bloco dentro de um bloco SEMPRE para que este teste seja feito o tempo todo. Depois coloque um controle que determine o início do Script. Observe o exemplo da figura ao lado. Duplica o bloco e troque “Gol 1” por “Gol 2” e “Ponto para o Jogador 2” por “Ponto para o Jogador 1”



**Exercício:** Faça com que ele se movimente para todos os lados e que fique mudando de traje

### 2.30. Abrindo projetos já existentes

Podemos ainda visualizar projetos já existentes.

Para isso, clicamos no menu arquivo → Abrir...

Clica-se em exemplos, então é só escolher o que desejar

### 2.31. Compartilhando

Para fazer o compartilhamento, deve-se colocar o login e a senha com que se registrou no Scratch e, também, o nome do projeto. Pode-se assinalar várias etiquetas, para que o projeto seja localizado de forma mais rápida. Depois podes também escrever algumas notas sobre o projeto, por exemplo instruções de utilização. Finalmente clicas em OK para enviá-lo.

## Capítulo 3. Python

---

### 3.1. Introdução ao Python

---

Devemos lembrar primeiramente, antes de entrar nesta linguagem de programação alguns conceitos importantes.

Temos o algoritmo que é um método para solucionar problemas. A partir do algoritmo é gerado o programa. Os programas tipicamente processam dados de entrada e produzem dados de saída.

### 3.2. O que é o Python

---

Python é uma linguagem de programação poderosa e de fácil aprendizado. Ela possui estruturas de dados de alto nível eficientes, bem como adota uma abordagem simples e efetiva para a programação orientada a objetos. Sua sintaxe elegante e tipagem dinâmica, em adição à sua natureza interpretada, tornam o Python ideal para *scripting* e para o desenvolvimento rápido de aplicações em diversas áreas e na maioria das plataformas.

O interpretador de Python e sua extensa biblioteca padrão estão disponíveis na forma de código fonte ou binário para a maioria das plataformas a partir do site, <http://www.python.org/>, e deve ser distribuídos livremente. No mesmo sítio estão disponíveis distribuições e referências para diversos módulos, programas, ferramentas e documentação adicional com a contribuição de terceiros.

O interpretador de Python é facilmente extensível incorporando novas funções e tipos de dados implementados em C ou C++ (ou qualquer outra linguagem acessível a partir de C). Python também se adequa como linguagem de extensão para customizar aplicações.

Esta apostila introduz o leitor informalmente aos conceitos básicos e aspectos do sistema e linguagem Python. É aconselhável ter um interpretador Python disponível para se poder “por a mão na massa”, porém todos os exemplos são auto-contidos, assim a apostila também pode ser lida sem que haja a necessidade de se estar *on-line*.

Para uma descrição dos módulos e objetos padrão, veja o documento Referência da Biblioteca Python. O Manual de Referência Python oferece uma definição formal da linguagem. Para se escrever extensões em C ou C++, leia **Estendendo e Embutindo o Interpretador Python** e **Manual de Referência da API Python/C**. Existem também diversos livros abordando Python em maior profundidade.

Esta apostila não almeja ser abrangente ou abordar todos os aspectos, nem mesmo todos os mais frequentes. Ao invés disso, ele introduz muitas das características dignas de nota em Python, e fornecerá a você uma boa ideia sobre o estilo e o sabor da linguagem. Após a leitura, você deve ser capaz de ler e escrever programas e módulos em Python, e estará pronto para aprender mais sobre os diversos módulos de biblioteca descritos na Referência da Biblioteca Python.

No final desta apostila há alguns *links* de referência, com os quais você deve acessar caso se interesse por saber mais sobre a linguagem Python.

### 3.3. Iniciando o uso do Python

---

Por Python ser uma linguagem de fácil entendimento, a sua curva de aprendizado é bastante curta. Inicialmente utilizaremos como base para o aprendizado de lógica de programação, mas Python pode ser usado em diversos *scripts* de automatização de tarefas básicas no computador, passar por aplicativos *desktops* com interfaces gráficas<sup>1</sup> e chegar a sistemas para internet bastante complexos.

### 3.4. Utilizando o Prompt do python

---

Vamos tentar utilizar alguns comandos básicos em Python. Quando o prompt estiver preparado para receber comandos ele vai lhe indicar com o prompt primário '>>>'. Em um outro momento iremos visualizar também o prompt secundário '...'. Sua função será vista mais a frente.

Vamos tentar alguns comandos simples em Python. Inicie o interpretador e aguarde o prompt primário, '>>>'. Agora vamos começar a executar comandos simples:

```
>>> 10
10
```

**Exercício: Execute vários numerais em seguida para começar a usar o prompt do Python.**

#### 3.4.1. Utilizando o Python como uma calculadora (utilizando números e operadores aritméticos simples)

---

O interpretador atua como uma calculadora bem simples: você pode digitar uma expressão e o valor resultante será apresentado após a avaliação da expressão. A sintaxe da expressão é a usual: operadores +, -, \* e / funcionam da mesma forma que em outras linguagens tradicionais (por exemplo, Pascal ou C):

Perceba a precedência dos operadores, se não utilizarmos parênteses no primeiro exemplo a multiplicação será realizada antes da subtração e o valor talvez não seja o esperado.

```
>>> 2-3*5
-13
>>> (2-3)*5
-5
>>> 12/3
4
```

**Exercício: Resolva algumas operações levando em consideração a precedência dos operadores e fazendo uso de parênteses para resolver os possíveis problemas.**

---

<sup>1</sup> Interface Gráfica são janelas que ajudam na usabilidade de um programa. O seu editor de textos é um programa com uma interface gráfica.



### 3.4.2. Inserindo comentários(#);

---

Podemos ainda fazer comentários com o caractere “#”, todo comentário é precedido deste caractere, pode ser utilizado tanto no início da linha como no meio da linha, logo após uma instrução válida. Quando fazemos um comentário no código, esta linha se torna imperceptível para o interpretador Python. Comentários como estes deixam seu código mais legível para outros programadores e ajudam no entendimento de certos trechos do seu programa.

```
>>> # Isso é um comentário.  
... 2+2  
4  
>>> 2+3 # Comentário também!  
5
```

No trecho em vermelho do código acima observamos o prompt secundário, comentado na seção 3.4. O prompt secundário está esperando alguma ação do programador (neste momento você pode incrementar seu código, mesmo estando no interpretador Python) seja uma simples equação ou “o pressionar do Enter”.

**Exercício: Escreva comentários no interpretador, mescle com comentários no meio da linha (após operações aritméticas).**

### 3.4.3. Tipo de variáveis / Tipos de Dados

---

Tipos de Dados são categorias de valores, onde dentro de cada categoria os dados (da categoria “X”) são processados de forma semelhante.

Por exemplo, números inteiros são processados de forma diferente dos números de ponto flutuante (decimais) e dos números complexos.

Tem-se tipos de dados primitivos que são aqueles já embutidos no núcleo da linguagem, onde normalmente são divididos entre simples e compostos.

Os simples são normalmente os números (*int*, *long*, *float*, *complex*) e cadeias de caracteres (*strings*).

Os compostos são por exemplo as listas, dicionários, tuplas e conjuntos.

Existem também os tipos definidos pelo usuário que são correspondentes a classes na orientação ao objeto.

#### Variáveis

As variáveis são nomes dados a áreas de memória onde os nomes podem ser compostos de algarismos, letras ou `_`; Porém o primeiro caractere não pode ser um algarismo; Deve-se ter o cuidado com o uso de palavras reservadas (*if*, *while*, etc) pois são proibidas usá-las como variável.

As variáveis servem para guardar valores intermediários, construir estruturas de dados.

Uma variável é modificada usando o comando de atribuição:

```
Var = expressão
```

É possível também atribuir valores a várias variáveis simultaneamente:

```
var1,var2,...,varN = expr1,expr2,...,exprN
```

#### 3.4.4. Atribuição de variáveis

---

Uma variável é modificada usando o comando de atribuição, no Python caracterizada pelo sinal de igual (=) .

Variáveis são criadas dinamicamente e destruídas quando não mais necessárias, por exemplo, quando saem do escopo de um bloco de instruções. O tipo de uma variável em Python muda conforme o valor atribuído. Se tratando do geral em termos de linguagens de programação temos alguns tipos de dados comuns a todas elas (*int, float, string*).

**Dica: Em algumas linguagens de programação o tipo da variável é definida no início do código, e esta variável só irá aceitar valores do tipo definido!**

Observe alguns exemplos:

```
>>> a = "1"
>>> b = 1
```

Observe que as duas variáveis acima foram declaradas com tipos diferentes, ao tentarmos somar os valores da variável “a” com a variável “b”, temos o seguinte erro:

```
>>> a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

O erro informa que não é possível concatenar (ou somar) dois tipos de dados diferentes. Neste exemplo são eles: *String* e *Inteiro*.

#### 3.4.5. Utilização de constantes e variáveis no Python

---

Podemos então também começar a atribuir valores a variáveis e visualizar os seus valores:

```
>>> a = 1
>>> a
1
>>> a = 2 * a
>>> a
2
>>> a, b = 3 * a, a
>>> a, b
(6, 2)
>>> a, b = b, a
>>> a, b
(2, 6)
```

### 3.4.6. Utilizando Strings

---

As linguagens de programação fornecem meios de manipular Strings. Python também trabalha com elas de diversas formas, elas podem ser delimitadas com aspas simples ou aspas duplas.

```
>>> 'string com aspas simples'
'string com aspas simples'

>>> s = "string com aspas duplas"
```

Podemos ainda trabalhar com uma sequência de *Strings*, para isso, fazemos uso de três (3) aspas (simples ou duplas) juntas. `'''` ou `"""`

```
>>> texto = """ Esse texto será impresso
                da maneira como foi
                escrito"""
```

Mais a frente veremos em um tópico separado, o verdadeiro uso deste outro tipo de *string* comentado acima.

### 3.4.7. Visualizando o tipo de variável trabalhada (`type()`)

---

Muitas vezes necessitamos saber o tipo de variável com a qual estamos trabalhando e para isso utilizamos o método `type()`. No exemplo a seguir, declaramos uma variável de nome “a” e atribuímos a ela o valor número 2 inteiro.

```
>>> a = 2
>>> type(a)
<type 'int'>
```

Ao mudarmos o valor da variável “a” para um outro tipo, como por exemplo, um valor real.

```
A = 2.0
type(a)
<type 'float'>
```

Verificamos com a função `type()`, que a modificação aconteceu sem erros. Esse resultado sem erros é uma particularidade da linguagem Python e outras linguagens de alto nível. Relembrar NOTA na seção 3.4.4.

**Exercício:** Declare outras variáveis e utilize o comando `type(variável)` para visualizar o tipo da mesma, procure utilizar variáveis de outros tipos que não Inteiros e de ponto flutuante.

### 3.4.8. Imprimindo resultados (`print()`)

---

Até o momento, ainda não imprimimos nenhuma informação na tela. Python nos fornece a função `print()` para resolver essa questão. A sua sintaxe é bem simples: `print(variavel)` (sem aspas). Com esta função, nós imprimimos todos os tipos de dados na saída padrão do computador, que é o monitor.

Supondo a declaração de uma variável, temos o seguinte exemplo:

```
>>> print(b)
2.0
>>> print("imprimindo uma string")
imprimindo uma string
```

Podemos formatar a saída da função `print()` com alguns parâmetros, para imprimirmos na tela as expressões:

```
a + b = 5
c + d = 7.4
Meu nome é João e eu tenho 21 anos.
```

Contando que o programador já tenha, anteriormente, declarado as variáveis, temos o seguinte resultado:

```
>>> print("a + b = %d" % (a+b))
a + b = 5
>>> print("c + d = %.1f" % (c+d))
c + d = 7.4
>>> print("Meu nome é %s e eu tenho %d anos." % (nome, idade))
Meu nome é João e eu tenho 21 anos.
```

Perceba a diferença entre uma instrução e outra (em vermelho), no primeiro caso `%d` indica que será impresso um valor inteiro, já no segundo caso o `%.1f` dará lugar a um valor de ponto flutuante com apenas uma casa decimal. Já no terceiro exemplo, temos a utilização de duas variáveis (uma *String* e outra do tipo inteiro) em uma mesma função `print()`.

Existem ainda outras formas de impressão formatada. Para cada tipo de dado, existe um símbolo correspondente, veja a tabela abaixo.

<code>%d</code>	Imprime um valor inteiro.
<code>%s</code>	Imprime uma variável do tipo <i>String</i> .
<code>%X</code>	Imprime a variável inteira na forma de Hexadecimal.
<code>%o</code>	Imprime a variável inteira na forma de Octal.
<code>%f / %.nf</code>	Imprime um valor de ponto flutuante / Imprime um valor de ponto flutuante com n casas decimais.

### 3.4.9. Utilizando a quebra de linha (`\n`)

No final de toda impressão utilizando a função `print()`, o cursor vai para uma nova linha. Isso se deve ao caractere especial (não impresso) "`\n`" (sem aspas), sua função é "quebrar a linha". Basicamente é equivalente à ao escrevermos uma linha no editor de textos e em seguida apertar o Enter.

Sua utilização explícita pode ser feita da seguinte maneira:

```
>>> var1 = "oi,"
>>> var2 = "\n"
>>> var3 = "tudo bem?"
>>> print("%s %s%s")
oi,
tudo bem?
```

Também é possível escrever constantes *String* em várias linhas incluindo as quebras de linha usando três aspas duplas ou simples como delimitadores, veja o exemplo:

```
>>> print """Projeto E-jovem
Lógica de Programação
Python"""
Projeto E-jovem
Lógica de Programação
Python
```

**Exercício:** Trabalhar com os tipos de variáveis no Python, tentando ver as possíveis formas de se utilizar a quebra de linha.

### 3.5. Trabalhando com bloco de instruções e fazendo um programa

Nos tópicos passados, trabalhamos apenas com o interpretador do Python, e com isso, toda vez que saíamos do interpretador, as variáveis utilizadas desapareciam, isso é, o trabalho era perdido. Se precisarmos criar um *script* para automatizar alguma tarefa no sistema, ou até mesmo um programa para calcular uma expressão matemática, precisamos lidar com arquivos contendo código Python. Todo arquivo Python, tem a extensão **.py**. Existe ainda algumas particularidades que serão comentadas a seguir.

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
```

É uma boa prática colocar estas duas linhas no início de arquivos Python. A primeira indica ao *script* onde está o interpretador Python no ambiente Linux. A segunda nos permite colocar caracteres acentuados nestes programas/scripts.

Podemos agora escrever o primeiro programa. Abra o editor de textos e digite o seguinte trecho de código:

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 # exemplo1.py
4
5 a = 26 b = 5
7 c = (a + b) * a
8
9 print("O valor de (2 + 5) * 2 é igual a %d" % c)
```

### 3.6. Rodando no shell o programa

---

Feito isso, devemos executar o programa no *shell* do Linux. Abriremos então o terminal do Linux. Pressione a combinação de teclas ALT+F2 e digite “konsole” (sem aspas). Dirija-se ao local onde foi salvo o arquivo **exemplo1.py** utilizando seu conhecimento em linux (utilize `cd /pasta`). Executamos o arquivo com o seguinte comando:

```
coordenador@cord06:~$ python exemplo1.py
O valor de (2 + 5) * 2 é igual a 14
```

Perceba que nosso programa não é nada complexo, com o tempo vamos incorporar mais código ao nosso exemplo e fazendo novos programas.

### 3.7. Docstrings(quando precisa-se inserir um comentário maior, como uma documentação);

---

Utilizamos *docstrings* quando precisamos inserir um comentário maior, como por exemplo, uma documentação, onde explicamos mais detalhadamente alguma parte de nosso código.

Nós iremos usar muito esta prática de documentação. Com ela iremos criar a documentação de um programa de forma mais profissional e prática.

Ao finalizarmos a tarefa de documentação do código, podemos gerar um arquivo **.html**, ou talvez um arquivo **.pdf**, para servir de estudo ou até mesmo para manutenção do código escrito. Porém esta particularidade não será vista nessa apostila.

Edite o arquivo **exemplo1.py** e deixe da seguinte forma.

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # exemplo1.py
4
5  ''' Este programa calcula a expressão
6      (a_b)*2 e imprime seu valor '''
7
8  a = 2
9  b = 5
10 c = (a + b) * a
11
12 print("O valor de (2 + 5) * 2 é igual a %d" %c)
```

Na seção 3.15 iremos ver seu uso de forma mais apropriada.

**Exercício:** Faça um programa para calcular a área de um triângulo. Documente seu programa para que qualquer pessoa possa fazer uso dele.

### 3.8. Trabalhando com a biblioteca ou módulo *math*

---

Python contém uma extensa lista de bibliotecas (ou módulos) que podemos usar em nossos programas. Temos bibliotecas para tratamentos de imagens, cálculos matemáticos, acessos ao sistema operacional, acesso a banco de dados, tratamento de áudio e vídeo, instruções básicas de jogos e também de comunicadores instantâneos, enter muitos outros. Podemos sempre usá-las juntas, isso é, o uso de um módulo não interfere o uso de um outro módulo.

Mais a frente aprenderemos como fazer nossa própria biblioteca, e então, seu colega poderá usar essa biblioteca, bastando somente ela está bem documentada para a utilização ser plena.

A instrução básica para manipular módulos é o *import*. Voltamos então ao interpretador Python e vamos importar o módulo *math*. O que acontece quando importamos um módulo é que ele contém algumas funções úteis e agora estarão disponíveis para o programador.

```
>>> import math
```

No módulo *math*, temos a função *pi* que nada mais é que o retorno da constante pi. Para termos este resultado fazemos o seguinte:

```
>>> math.pi
3.1415926535897931
```

Uma outra maneira de importar módulos ou apenas algumas funções daquele respectivo módulo é utilizar o comando *from*, por exemplo: No módulo *math*, o qual acabamos de importar existe a função *sqrt()*, que nos retorna a raiz quadrada de um número. Demonstramos seu uso a seguir:

```
>>> from math import sqrt
>>> sqrt(4)
2.0
```

A seguir segue uma tabela com as funções mais básicas do módulo *math*.

<code>math.sqrt(x)</code>	Raiz Quadrada do valor <b>x</b> .
<code>math.cos(a)</code>	Cosseno do ângulo <b>a</b> *
<code>math.sin(a)</code>	Seno do ângulo <b>a</b> *
<code>math.tan(a)</code>	Tangente do ângulo <b>a</b> *
<code>math.radians(a)</code>	Converte o ângulo <b>a</b> de graus para radianos
<code>math.pi</code>	Informa a constante pi
<code>math.hypot(x,y)</code>	Calcula a hipotenusa de um triângulo retângulo
<code>math.pow(x,y)</code>	Eleva o número <b>x</b> a potência <b>y</b>

\* Python trabalha com ângulos em radianos, nós usualmente utilizamos ângulos em graus. Para utilizarmos as funções de trigonometria precisamos transformar os ângulos radianos para graus utilizando a função (`math.radians()`).

Python tem a função `pow(x, y)` que eleva um número a uma potência, mas nós também temos um operador matemático que nos auxilia nesta tarefa. Querendo calcular  $2^3$ , normalmente faríamos `a = 2 * 2 * 2` ou `pow(2,3)`, porém se quisermos calcular  $2^{21}$  fica difícil escrevermos `a = 2 * 2 * 2 * 2 * 2 * 2 * 2...` tantas vezes. Para resolver esse problema podemos escrever:

```
>>> a = 2 ** 3
>>> print a
8
>>> a = 2 ** 10
>>> print a
1024
```

Para uma lista completa das funções do módulo *math* e de suas respectivas descrições leia a documentação<sup>2</sup> (em inglês).

<sup>2</sup> A documentação completa da versão atual pode ser encontrada no sítio <http://docs.python.org/>, uma versão em português, porém não completa pode ser encontrada em:

**Exercício:** Utilize estas funções para calcular valores de suas aulas de matemática.

### 3.9. Mais string:

---

Na seção 3.4.7, falamos pela primeira vez sobre *Strings* e sua utilização foi bem simples. Agora vamos aprender a acessar e “atualizar” certas partes da *String*, melhorar a formatação de saída fazendo uso de caracteres especiais e utilizar algumas funções que fazem o tratamento de *Strings*.

```
>>> texto = "Mundo e-Jovem"
>>> print texto[0]
M
>>> # Imprimindo toda uma palavra, do caractere 0 ao caractere 4
>>> print texto[0:5]
Mundo
```

Ilustrando:

M	u	n	d	o		e	-	J	o	v	e	m
0	1	2	3	4	5	6	7	8	9	10	11	12

OBS: Podemos ver que o primeiro caractere da *String* texto tem como índice o número zero (0), esta convenção é bastante utilizada em diversas outras linguagens de programação.

#### 3.9.1. Operadores e Strings

---

O tipo de dado *String* aceita que os programadores façam certas operações com seus “componentes”, como por exemplo, concatenar, ou repetir seus elementos.

A função `len(variável)` que é padrão no Python, é muito utilizada em *Strings*, pois com ela podemos saber o tamanho da *String*, isso é, saber quantos caracteres ela contém, incluindo seus espaços.

```
>>> len(texto)
13
```

No tópico 3.9, fizemos uso do `import` para importar o módulo *math*, no caso de strings não precisamos importar a biblioteca referente (*str*), pois o interpretador já resolve isso pra nós quando é iniciado.

Quando trabalhamos com programas simples, geralmente feitos sem interface gráfica, podemos incrementar o visual com tabulações e linhas. Facilmente podemos mostrar isso na tela.

```
>>> print '-' * 20
-----
```

#### 3.9.2. Alinhamento de Strings

---

Podemos também ajustar um texto na tela utilizando as funções (`ljust()`, `rjust()` e `center()`). Em cada um dos exemplos abaixo, digitamos o valor 50, este é o número de espaços que a frase irá pular. No caso do alinhamento à direita, ela deixa 50 espaços em branco até o primeiro caractere da *String*. No texto centralizado ocorre uma divisão e fica metade dos caracteres à esquerda e outra metade à direita.



```
>>> esquerda = 'Texto alinhado a esquerda'
>>> centralizado = 'Texto centralizado'
>>> direita = 'Texto alinhado a direita'

>>> print(esquerda.ljust(50))
Texto alinhado a esquerda

>>> # Obtivemos mudanças?

>>> print(centralizado.center(50))
        Texto centralizado

>>> print(direita.rjust(50))
                Texto alinhado a direita
```

Perceba que o alinhamento à esquerda não aconteceu. Isso porque o padrão de saída da função `print()` é alinhado à esquerda. Então para que utilizamos esta função? Porque podemos ainda incrementar a função com um caractere a ser impresso. Perceba que é **apenas um caractere** – não podendo ser mais de um. Observe:

```
>>> print(esquerda.ljust(50, '+'))

Texto alinhado a esquerda+++++++

>>> print(centralizado.center(50, '-'))
-----Texto centralizado-----
```

Fora estes tratamentos com relação ao alinhamento, podemos ainda trabalhar mais próximos dos caracteres, como por exemplo, modificando a *String* para ser exibida totalmente em minúscula ou em maiúsculas por completo, ou ainda em outros formatos.

```
>>> var1 = 'TeXto SimpLes'
>>> print(var1.lower())
texto simples
>>> var1 = 'texto simples'
>>> print(var1.upper())
TEXTO SIMPLES
>>> var1 = 'texto simples'
>>> print(var1.title) # Deixa maiúscula apenas o 1º caractere de cada palavra
Texto Simples
>>> var1 = 'tEXTO SIMPLES'
print(var1.capitalize())
Texto simples # Deixa maiúscula apenas o 1º caractere da String
```

Com algumas ferramentas e um pouco de criatividade podemos criar um menu bem simples para um possível programa em modo texto.

**Exercício: Programe um menu utilizando as funções para tratamento de *strings* para que ele quando executado seja impresso da seguinte maneira:**

```
-----MENU-----
      1 - Opção 1
      2 - Opção 2
      3 - Opção 3
-----2009-----
```

### 3.10. Entrada de dados (*raw\_input*);

A medida que nossos programas vão crescendo, e nós vamos adicionando funcionalidades a ele, eles vão se tornando mais complexos e precisando muitas vezes, que nós entremos com algum dado, sendo um nome para um cadastro ou um ângulo para o cálculo do seu cosseno.

Para isso nós utilizamos a função *raw\_input()*. Esta função retorna uma *String*, sua utilização é da seguinte maneira:

```
>>> a = raw_input('Digite seu nome: ')
Digite seu nome:

>>> print a
João
```

Já que a função *raw\_input()* retorna uma *String*, qual método utilizamos para receber um valor inteiro, ou um valor real?

Para isso fazemos uso da conversão de tipos. Em Python, nós podemos converter quase todo tipo de dado em qualquer outro tipo de dado, seja de **Caractere para Inteiro** ou de **Inteiro para Complexo**. Utilizamos juntamente algumas outras funções para recebermos um dado requerido. Basta pegarmos o tipo de dado e usarmos como uma função.

Digamos que o valor que você espera receber é um número real, isso é, *float* em Python. Então utilizamos a função *float()* da seguinte maneira:

```
>>> a = float(raw_input('Digite um numero: '))
Digite um numero:

>>> print a
>>> 2.0
```

Com esse artifício, a variável “a” já tem um valor do tipo real (*float*). Esta técnica pode ser usada para obter todos os outros tipos de dados em Python. Segue uma tabela com alguns tipos de dados. Para uma lista completa verifique a documentação.

<code>int(X)</code>	Converte X para um inteiro
<code>float(X)</code>	Converte X para um valor de ponto flutuante (real)
<code>complex(X, [Y]*)</code>	Converte X para um valor real e Y para um valor imaginário: X+Yj
<code>str(X)</code>	Converte X para uma String
<code>chr(X)</code>	Converte X (sendo X inteiro) para um caractere
<code>ord(X)</code>	Converte um único caractere para seu valor inteiro correspondente
<code>hex(X)</code>	Converte X (inteiro) para uma String representando seu valor em Hexadecimal
<code>oct(X)</code>	Converte X (sendo X inteiro) para uma String representando seu valor em Octal

\* Os colchetes na função *complex(X, [Y])*, indicam que o valor a ser passado para a função, no caso o parâmetro Y, é optativo. Pode ou não ser passado e o resultado da chamada da função será válida.

**Exercício:** Faça com que o valor que *raw\_input()* retorne, seja um inteiro, um número complexo e um valor de *String* (hexadecimal e octal incluso). Após o exercício, verifique se está correto utilizando a função *type(variável)*.

### 3.11. Operadores Aritméticos e Expressões Lógicas

---

Nós estamos utilizando em nossos exemplos e exercícios, operadores aritméticos básicos (+, -, \*, /), neste tópico veremos alguns adicionais que são necessários, para uma programação mais rápida e eficiente. São eles:

%	Módulo: Retorna o resto da divisão entre dois números
**	Exponenciação: Eleva um número a uma potência*
//	Floor Division: Divide um número por outro, porém a casa decimal é retirada

\*Visto na seção 3.8.

Levando em consideração que  $a = 10$  e  $b = 20$ , temos os seguintes exemplos:

```
>>> b % a
0.0
>>> a ** 32 # 1032
(número muito grande)
>>> 9.0 // 2.0
4.0
>>> pow(2, 3)
8
```

Perceba que o valor da divisão  $9.0 / 2.0$  é igual a 4.5, porém o operador (//) retira a parte decimal do resultado.

Importando o módulo *math* e trabalhando com algumas expressões:

```
>>> #calculando o valor do volume de uma esfera com raio = 5
>>>
>>> volume = (4.0 / 3.0) * math.pi * (5**3)
>>> print volume
523.59877559...
>>>
>>> #calculando o valor da expressão:  $2^2 + \sqrt{(\exp^2 + 1)}$ 
>>>
>>> y = (2**2) + math.sqrt(math.exp(2) + 1)
>>> print y
6.8963867...
```

### 3.12. Operadores Lógicos e Operadores Relacionais

---

Como visto no tópico 1.14, operadores Relacionais e Lógicos são usados quando precisamos executar um teste dentro de um laço, ou para causar algum desvio dentro do programa. Com estes operadores o programador tem agora um conhecimento maior para incrementar seu código e também para resolver problemas básicos de programação.

### 3.12.1. Trabalhando no shell com operadores lógicos

---

Considerando  $a = 10$  e  $b = 20$ , temos os seguintes exemplos:

```
>>> a > b
False
>>> a < b
True
>>> a == b
False
>>> a >= 10
True
>>> b <= a
False
>>> a != b
True
```

**Exercício:** Utilize estes operadores relacionais juntamente com outros tipos de dados, como uma *String* e tipo *Complex*.

De acordo com o que foi visto no tópico 1.5, Python também trás os operadores lógicos e nós podemos fazer as respectivas comparações de acordo com a sintaxe.

Considerando  $a = \text{True}$  e  $b = \text{False}$ , verifique com a função `type()` qual o tipo destas variáveis, veja os seguintes exemplos:

```
>>> type(a)
<type 'bool'>
>>> type(b)
<type 'bool'>
>>> a and b
False
>>> a or b
True
>>> not(a and b)
True
```

**Exercício:** Faça testes lógicos para se acostumar com o uso desse tipo de operação.

### 3.12.2. Expressões Lógicas

---

Podemos testar se uma expressão terá como resultado um valor verdadeiro ou falso utilizando operadores lógicos.

Supondo  $a = \text{True}$ ,  $b = \text{False}$  e  $c = \text{True}$  temos algumas possibilidades.

```
>>> not(a and b) or b
True
>>> (a or b) and b
False
>>> (a and b and c) or c
True
>>> (a and b and c) or not(c)
False
```

### 3.13. Estrutura de Seleção(Decisão ou Condição – if)

---

Construções condicionais são utilizados para incorporar a tomada de decisões em programas. O resultado desta decisão determina a sequência em que um programa irá executar as instruções. Você pode controlar o fluxo de um programa usando construções condicionais.

A declaração *if* de Python é semelhante a de outras linguagens. Ela contém uma expressão lógica - uma condição - com que os dados são comparados e uma decisão é tomada com base no resultado da comparação.

```
>>> if <condição>:  
...     <bloco de comandos>
```

Na instrução acima, a <condição> é testada, se for diferente de 0 (zero), ou um valor booleano Verdadeiro, o <bloco de comandos> será executado, caso contrário, a primeira instrução após o bloco de comandos é executado.

Dica: Novamente aqui se nota a aparição do prompt secundário (visto na seção 3.4), neste momento você poderá digitar uma sequência de comandos e o interpretador espera que o programador digite algo.

#### 3.13.1. Seleção Composta (if..else)

---

O comando *if* poderá ser usado juntamente com um componente chamado *else*. Seu uso segue a mesma linha da instrução *if*.

```
>>> if <condição>:  
...     <bloco de comandos>  
... else:  
...     <bloco de comandos 2>
```

O <bloco de comandos 2> só será executado, se a condição for igual 0 (zero) ou tiver um valor booleano Falso.

A declaração *else* é opcional e pode haver no máximo uma declaração *else* seguindo uma instrução *if*.

Exemplo de um uso real:

```
1  #!/usr/bin/env python  
2  # -*- coding: utf-8 -*-  
3  
4  # Testando se um número é impar ou par.  
5  
6  num = float(raw_input("Digite um número: "))  
7  if (num % 2 == 0):  
8      print("número %.1f é par" % num)  
9  else:  
10     print("número %.1f é impar" % num)
```

### 3.13.2. Seleção Encadeada ou Decisão Aninhada

A declaração *elif* permite verificar múltiplas expressões de valor verdadeiro ou falso e executar um bloco de código, logo que uma das condições é avaliada como verdadeira.

```
>>> if <condição 1>:  
... <bloco de comandos 1>  
... elif <condição 2>:  
... <bloco de comandos 2>
```

Como o *else*, a declaração *elif* é opcional. No entanto, ao contrário da outra declaração, para o qual não pode haver mais de uma aparição, pode haver um número arbitrário de instruções *elif* na sequência de um caso.

Exemplo de uso real:

```
1 #!/usr/bin/env python  
2 # -*- coding: utf-8 -*-  
3  
4 """ Programa que pergunta em que turno você estuda.  
5     Recebe um valor (M - Matutino, V - Vespertino ou N - Noturno)  
6     e imprime a mensagem "Bom dia!", "Boa Tarde!"  
7     ou "Boa Noite!" """  
8  
9 turno = raw_input("Em que turno você estuda? (Digite, M, V ou N)  
\n>>>")  
10 if (turno.upper() == 'M'):  
11     print("Bom Dia!")  
12 elif (turno.upper() == 'V'):  
13     print("Boa Tarde!")  
14 elif (turno.upper() == 'N'):  
15     print("Boa Noite!")  
16 else:  
17     print("Escolha Inválida.")
```

### 3.13.3. A construção *if..elif..else* aninhados

Pode haver uma situação em que você necessite verificar uma outra condição dentro de uma condição. Nessas situações podemos aninhar declarações *if* sem problemas.

Em uma Construção desse tipo, você pode ter um *if..elif..else* dentro de um outro *if..elif..else* e assim por diante.

Sua sintaxe a seguir:

```
>>> if <condição 1>:  
... <bloco de comandos 1>  
... if <condição 2>:  
...     <bloco de comandos 2>  
...     elif <condição 3>:  
...     <bloco de comandos 3>  
... else  
...     <bloco de comandos 4>  
... elif <condição 5>:  
... <bloco de comandos 5>  
... else:  
... <bloco de comandos 6>
```

Segue um exemplo real:

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  var = 100
5  if var < 200:
6      print "O valor da expressão é menor que 200."
7      if var == 150:
8          print "Que é igual a 150."
9      elif var == 100:
10         print "Que é igual a 100."
11     elif var == 50:
12         print "Que é igual a 50."
13 elif var < 50:
14     print "Valor da expressão é menor que 50."
15 else:
16     print "Nenhuma expressão é verdadeira."
```

**Exercício:** Faça um Programa que verifique se uma letra digitada é vogal ou consoante.

### 3.14. Controle de escopo por indentação

---

Em nossos exemplos didáticos e também nos reais, a declaração *if* <condição>: está mais a esquerda da declaração <bloco de comandos>. Chamamos de bloco indentado aquele que está mais internamente a uma instrução superior. Por Python não ter caracteres indicando o fim de uma linha ou o início e término de um bloco, é desta maneira simples que ele deixa o código organizado e de fácil entendimento posterior pelo programador. Portanto, em Python, não é somente uma boa prática indentar seu código, como também é obrigatório. Assim, qualquer programador que for ler seu código irá entender sem grandes complicações.

### 3.15. Criando Funções

---

Uma função é um bloco de código, organizado reutilizável que é usado para executar uma ação única. As funções proporcionam uma melhor modularidade para seu aplicativo e um alto grau de reutilização de código.

Como você já sabe, Python oferece muitas funções padrões como `print()` e `raw_input()`, entre outros, mas você também pode criar suas próprias funções. Essas funções são chamadas funções definidas pelo usuário.

#### 3.15.1. Definindo uma Função

---

Você pode definir funções para fornecer a funcionalidade necessária. Aqui estão as regras simples de definir uma função em Python:

- Blocos de função começam com a palavra-chave `def` seguida do nome da função e um par de parênteses em seguida `()` e após os parênteses, fechamos a linha com o caractere dois-pontos `:`. Esta linha chama-se assinatura da função;
- Todos os parâmetros de entrada ou argumentos devem ser colocados dentro desses parênteses. Você também pode definir estes parâmetros dentro de parênteses;

- A primeira declaração de uma função pode ser uma *DocString* (opcional);
- O bloco de código dentro de cada função começa indentado em relação a linha de assinatura;
- A instrução `return [expressão]` sai de uma função, opcionalmente, passando para quem chamou a função (geralmente uma variável) um valor. A instrução de retorno com nenhum argumento é o mesmo que nenhum retorno ou `return None`.

Pela sintaxe, temos o seguinte trecho de código:

```
1 #/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 def nome-da-funcao(parametro1, parametro2):
5     """ Documentação da funcionalidade da função, geralmente o que
6         ela realiza ao ser chamada """
7
8     <bloco_de_comandos>
9     return expressão
```

Por padrão, os parâmetros têm um comportamento de posicionamento, e você precisa informá-los na mesma ordem em que foram definidos.

A seguir temos uma forma bem simples de uma função do Python. Essa função recebe uma *String* como parâmetro de entrada e imprime na tela. Salve este trecho de código como **teste.py**

```
1 #/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 def imprimeString(frase):
5     """ Essa função recebe uma String e imprime ela na tela """
6     print frase
7     return
```

Definir uma função só dá a ela um nome e especifica os parâmetros que devem ser incluídas na função e as estruturas dos blocos de código.

Uma vez que a estrutura básica de uma função é finalizada, você pode executá-la, chamando-o de outra função ou diretamente a partir do prompt do Python.

```
>>> imprimeString('Imprima essa frase')
Imprima essa frase
>>> a = 'Isto é uma String'
>>> imprimeString(a)
Isto é uma String
```

### 3.15.2. Escopo de Variáveis

As variáveis de um programa podem ser restritas a certas partes deste mesmo programa. Isso depende de onde você tenha declarado uma variável.

O escopo de uma variável determina a porção do programa onde você pode acessar um identificador específico. Existem dois âmbitos básicos de variáveis em Python:

- Variáveis Globais
- Variáveis Locais



Variáveis que são definidas dentro de um corpo de uma função, só poderão ser usada em âmbito local, e aqueles definidos fora têm um alcance global.

Isto significa que as variáveis locais podem ser acessadas somente dentro da função em que são declaradas, ao passo que as variáveis globais podem ser acessadas por todo o corpo do programa e por todas as funções. Quando você chamar uma função, as variáveis declaradas dentro dela são trazidos para o escopo, ao término do uso desta função as variáveis deixam de existir na memória. A seguir temos um exemplo:

```
>>> def subtrai(a, b):  
...   c = a - b  
...   # c é uma variável local para a função subtrai  
...   return c
```

### 3.15.3. Argumentos Requeridos

---

Argumentos obrigatórios são os argumentos passados para uma função na ordem de posicionamento correto. O número de argumentos na chamada de função deve corresponder exatamente com a definição da função.

Ao chamarmos uma função como a nossa `imprimeString()`, você definitivamente precisa passar um argumento, do contrário, daria um erro de sintaxe, como a seguir:

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: imprimeString() takes exactly 1 argument (0 given)
```

### 3.15.4. Argumento Padrão

---

Um argumento padrão é um argumento que assume um valor padrão se o valor não está previsto na chamada de função para esse argumento.

Atualizando o arquivo `teste.py`, com a adição de mais uma função:

```
8   # (continuando)  
9   def somar(a, b = 1):  
10      """ Esta função soma dois números """  
11      c = a + b  
12      # a variável 'c' é uma variável local à função somar  
13      return c
```

Ao chamarmos esta nova função `somar()`, se não passarmos o parâmetro `b`, este não irá gerar o erro de falta de parâmetro. A função vai assumir que o valor de `b` será 1.

```
>>> somar(2)  
3  
>>> somar(2, 2)  
4
```

### 3.15.5. Criando e utilizando seu próprio módulo

---

Após atualizarmos o arquivo **teste.py** com as duas funções criadas anteriormente, nós podemos utilizar este arquivo como um módulo de mesmo modo quando utilizamos o módulo *math*. Faça o seguinte:

1. Vá ao diretório onde o arquivo **teste.py** está salvo;
2. Chame o interpretador Python;
3. Digite: `import teste` (perceba que não precisa da extensão `.py`);
4. Inicie o uso do módulo;

```
>>> import teste
>>> teste.imprimeString('Utilizando o meu primeiro módulo')
Utilizando o meu primeiro módulo
>>> teste.somar(4, 6)
10
```

## 3.16. Listas

---

### 3.16.1. Introdução

---

A mais básica estrutura de dados em Python é a sequência. A cada elemento de uma sequência é atribuído um número, a sua posição, ou índice. O primeiro índice é zero, o índice 1 (um) é a segunda posição, e assim por diante.

O Python tem seis tipos internos de sequências, mas os mais comuns são as listas, tuplas e dicionários. Nesta seção iremos ver o tipo Lista.

Há certas coisas que você pode fazer com todos os tipos de sequências. Essas operações incluem a indexação, cortar, adicionar, deletar, multiplicar, e verificar a adesão. Além disso, o Python tem funções embutidas para encontrar o comprimento de uma sequência, e para encontrar os seus elementos maior e menor, entre outros métodos menos utilizados.

Esse tipo de estrutura básica em outras linguagens é geralmente chamada de vetor ou *array*.

### 3.16.2. Listas

---

A Lista é um dos tipos de dados mais versáteis disponível em Python, que pode ser escrita como uma lista de valores separados por vírgulas (itens) entre colchetes. Uma boa coisa sobre as Listas de itens é que em uma Lista não precisa ter todos os elementos de mesmo tipo, isso é, em uma mesma Lista é possível ter elementos do tipo *float*, do tipo *int* e do tipo *String*.

Criar uma lista é tão simples como colocar valores separados por vírgulas entre colchetes. Por exemplo:

```
>>> lista1 = ['Física', 'Química', 2008, 2000]
>>> lista2 = [1, 2, 3, 4, 5 ]
>>> lista3 = ["a", "b", "c", "d"]
>>> lista4 = []
```

### 3.16.3. Acessando os valores de uma Listas

Para acessar os valores da Lista, usa-se o nome da variável da Lista juntamente com os colchetes, e internamente o índice ou índices (válidos) para obter um valor disponível nesta Lista. Existe ainda a possibilidade de trabalhar com índices negativos, neste caso, a contagem acontece da direita para esquerda. O último elemento tem como índice o valor -1, o penúltimo -2, e assim sucessivamente.

```
>>> print "primeiro elemento da lista1: ", lista1[0]
primeiro elemento da lista: Física
>>> print "ultimo elemento da lista1: ", lista1[-1]
ultimo elemento da lista1: 2000
```

Podemos também acessar um conjunto seguido de elementos da Lista:

```
>>> print "Do segundo ao último elemento da lista3: ", lista3[1:]
Do segundo ao último elemento da lista3: ['b', 'c', 'd']
```

Da seguinte maneira, acessamos um intervalo de valores dentro da Lista:

```
>>> print "Do segundo ao quarto elemento da lista2: ", lista2[1:3]
Do segundo ao quarto elemento da lista2: ['2','3','4']
```

### 3.16.4. Operações básicas da Lista

Listas respondem aos operadores + e \* muito parecido como as Strings respondem, aqui também os operadores significam concatenação e repetição respectivamente, exceto que o resultado quase sempre é uma nova lista, não uma sequência de caracteres.

```
>>> l1 = ['Oi!']
>>> l2 = ['Tudo bem?']
>>> l3 = l1 + l2
>>> # Concatenamos l1 a l2, gerando uma nova lista l3
>>> print l3
['Oi', 'Tudo bem?']
```

Podemos utilizar a multiplicação para repetir os elementos que estão dentro da lista. Ao escolhermos apenas 1 elemento da lista, sua multiplicação retornará uma *String* daquele único elemento multiplicado.

```
>>> l4 = l1 * 2
['Oi!', 'Oi!']
>>> l4[1] * 2
'Oi!Oi!'
```

### 3.16.5. Atualizando os valores da Lista

Você pode atualizar um ou vários elementos da lista, dando o valor ou a fatia dentro do colchete e atribuir a Lista um novo elemento. Python também permite você adicionar elementos **ao final** de uma Lista com o método *append()*\*:

```
>>> l1 = [1, 'a', 2, 'b', '2']
>>> # último elemento será atualizado para valor 3
>>> l1[4] = 3
>>> print l1
[1, 'a', 2, 'b', 3]
```

\* Discutiremos o método *append()* mais a frente.

### 3.16.6. Deletando valores da Lista

Para remover um elemento da lista, você pode usar o comando *del* se você sabe exatamente qual elemento deseja excluir ou o método `remove()`\* se você não conhece sua posição na Lista.

```
>>> l1 = ['Física', 'Química', 2008, 2009]
>>> # removendo o primeiro elemento da lista
>>> del l1[0]
>>> print l1
['Química', 2008, 2009]
```

\*Discutiremos o método `remove()` mais a frente.

Perceba que agora o primeiro elemento da lista é “Química”, portanto, seu índice referente é zero.

Python para facilitar o uso de listas, desenvolveu uma série de métodos que podem ser usados, de acordo com sua sintaxe, são estes os mais utilizados:

<code>lista.append(X)</code>	Adiciona o elemento X ao fim da lista
<code>lista.insert(i, X)</code>	Adiciona o elemento X na posição i
<code>lista.pop()</code>	Remove e retorna o último elemento da lista
<code>lista.remove(X)</code>	Remove o elemento X da lista
<code>lista.sort()</code>	Organiza a lista (parecido com a organização que um dicionário leva em conta)
<code>lista.reverse()</code>	Inverte a ordem da lista

## 3.17. Estrutura de Repetição

Um laço é uma construção que faz com que uma parte de um programa passe a ser repetido um certo número de vezes. A repetição continua enquanto a condição estabelecida para o ciclo continua a ser verdadeira. Quando a condição se torna falsa, o laço termina e o controle do programa é passado para a instrução seguinte ao laço.

## 3.18. Estrutura de repetição FOR;

O laço `for` em Python tem a capacidade de iterar sobre os itens de uma sequência, como uma lista ou uma *String*. A sua sintaxe é bem simples:

```
>>> for x in <intervalo>:
...     <bloco de instruções>
```

No lugar de `x`, indicamos a variável que irá controlar o ciclo. A variável vai tomar todos os valores do intervalo especificado, e a cada tomada, vai executar as instruções indicadas dentro do ciclo, ou seja, as que estão indentadas.

No lugar de `intervalo`, indicamos os valores que a variável irá tomar. Aqui podemos especificar um intervalo de números, uma lista, entre outros.

A seguir temos um exemplo da instrução `for()` utilizando como iterador uma lista de 5 números.

```
>>> lista = [1,2,3,4,5]
>>> for x in lista:
...     print x**2
...
1
4
9
16
25
```

No caso, o comando “for” verifica a lista e atribui a variável `x` o primeiro valor da lista, neste exemplo o valor 1, então executa os comandos identados que simbolizam um único bloco de comandos. Outro exemplo seguindo a mesma linha, porém ao invés de utilizarmos uma Lista, o objeto iterador é uma sequência de caracteres ou *String*.

```
>>> nome = 'Jovem'
>>> for letra in nome:
...     print "Letra Atual: ", letra
Letra Atual: J
Letra Atual: o
Letra Atual: v
Letra Atual: e
Letra Atual: m
```

Lembramos da seção 3.10 onde falamos mais sobre Strings que o primeiro caractere de uma *String* era referenciado por um índice, ao usarmos o comando `for`, ele vai utilizar este mesmo índice para se “locomover” dentro da *String*.

### 3.18.1. Função `range()`

Retorna uma lista contendo uma progressão aritmética de inteiros, isto é, `range (i, j)` retorna `[i, i +1, i +2, ..., j-1]`;

Se chamarmos a função da seguinte maneira:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lembram da seção 3.15.4 onde nós passamos para a função somente um parâmetro enquanto o outro assumia um valor padrão? Nesta função também fazemos uso deste recurso, o primeiro parâmetro assume valor 0 (zero) ao ser omitido. Segue o seguinte exemplo:

```
>>> for i in range(0,5):
...     print '%d elevado a 2: %d' % (i, i**2)
...
0 elevado a 2: 0
1 elevado a 2: 1
2 elevado a 2: 4
3 elevado a 2: 9
4 elevado a 2: 16
```

A instrução acima passa para a variável `i` o valor zero, executa o bloco de código, que neste exemplo é apenas o comando `print`, itera o valor de `i` para 1 e executa o bloco de código e assim sucessivamente.

Outro fato importante, é que o ciclo *for*, por padrão, atualiza a variável em uma unidade em cada iteração. Mas e se quisermos atualizar em 2 ou 3, 1.5 ou ainda 0.1? Ai acrescentamos mais um parâmetro ao `range()`. Você já deve ter notado que os dois valores do `range()` indicam o valor inicial e o valor final da iteração do ciclo. Então o terceiro valor que iremos acrescentar, indicará o valor da iteração. Veja o exemplo a seguir:

```
>>> for numero in range(0,15,2):
...   print 'os números pares são: ', numero
...
os números pares do intervalo são: 0
os números pares do intervalo são: 2
os números pares do intervalo são: 4
os números pares do intervalo são: 6
os números pares do intervalo são: 8
os números pares do intervalo são: 10
os números pares do intervalo são: 12
os números pares do intervalo são: 14
```

**Exercício:** Desenvolva um gerador de tabuada, capaz de gerar a tabuada de um número inteiro entre 1 a 10. O usuário deve informar de qual numero ele deseja ver a tabuada. A saída deve ser conforme o exemplo abaixo:

```
Tabuada de 5:

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
...
5 x 10 = 50
```

### 3.19. Comando de Repetição *while*

---

O ciclo *while* é uma das construções de repetição disponíveis em Python. O laço *while* continua até que a expressão se torne falsa. A condição tem de ser uma expressão lógica e deve retornar um valor verdadeiro ou falso. A seguir segue sua sintaxe:

```
>>> while <condição>:
...   <bloco de instruções>
```

Aqui a declaração <condição> é avaliada primeiro. Se a expressão é verdade, então o <bloco de instruções> é executado repetidamente até que a expressão se torne falsa. Caso contrário, a próxima instrução após o bloco de instruções é executado. Segue um exemplo simples:

```
>>> contador = 0
>>> while (contador < 9):
...     print 'O contador está em: ', contador
...     contador = contador + 1
...
O contador está em:  0
O contador está em:  1
O contador está em:  2
O contador está em:  3
O contador está em:  4
O contador está em:  5
O contador está em:  6
O contador está em:  7
O contador está em:  8
```

O bloco acima consiste na impressão e nas declarações de incremento. Ele é executado repetidamente até que a contagem seja inferior a 9. Com cada iteração, o valor atual da contagem do índice é apresentado e, em seguida, incrementado em 1. Experimente modificar a condição deste código para menor igual (<=) e verifique o resultado.

### 3.19.1. O Laço Infinito

---

Você deve ter cuidado ao usar laços *while*, devido à possibilidade de que esta condição nunca chegue a um valor falso. Isso resulta em um ciclo que não termina nunca. Esse laço é chamado de laço infinito. Um laço infinito pode vir a ser útil na Programação Cliente/Servidor onde o servidor precisa ser executado de forma contínua para que os programas clientes possam se comunicar com ele como e quando necessário. Exemplificando:

```
>>> var = True
>>> contador = 0
>>> while (var):
...     print "contador: ", contador
...     contador = contador + 1
...
contador: 0
contador: 1
contador: 2
contador: 3
contador: 4
contador: 5
(omitimos algumas saídas)
contador: 9851
```

Para cancelar o laço infinito pressione CTRL+C.

### 3.19.2. A declaração *break*

---

A instrução *break* em Python encerra o ciclo atual e continua a execução na instrução seguinte ao bloco de instruções pertencentes ao laço, assim como a quebra dos tradicionais laços encontrados em linguagem C.

O uso mais comum para a instrução *break* é quando alguma condição externa é disparada exigindo uma saída precipitada de um laço. A instrução *break* pode ser usado em ambos os ciclos *while* e *for*. Temos o seguinte exemplo:

```
>>> for letras in 'E-Jovem':
...     if letras == 'v':
...         break
...     print 'caractere atual: ', letras
...
caractere atual: E
caractere atual: -
caractere atual: J
caractere atual: o
```

### 3.19.3. A declaração *continue*

---

A instrução *continue* em Python retorna o controle para o início do laço *while* ou *for*. A instrução *continue* rejeita todas as instruções restantes na iteração atual do laço e move o controle de volta para o topo do laço. Utilizando o mesmo exemplo acima, temos:

```
>>> for letras in 'E-Jovem':
...     if letras == 'v':
...         print 'Estamos voltando ao topo do laço' # pulamos o caractere
...         'v' no exemplo
...         continue
...     print 'caractere atual: ', letras
...
caractere atual: E
caractere atual: -
caractere atual: J
caractere atual: o
Estamos voltando ao topo do laço
caractere atual: e
caractere atual: m
```

Diferentemente da declaração *break* que sai imediatamente do laço, o comando *continue* apenas pula uma iteração, continuando logo após sua execução ter sido resolvida.



### 3.19.4. A declaração *pass*

---

A declaração *pass* em Python é utilizada quando é necessária uma declaração sintaticamente, mas você não precisa que nenhum comando ou código seja executado.

A declaração de passagem é uma operação nula, nada acontece quando ela é executada. O comando *pass* também é útil em ambientes de desenvolvimento onde seu código provavelmente acabará indo, mas o trecho referente ainda não foi escrito. Vamos agora utilizar o exemplo anterior modificando a linha 3 para o comando *pass*.

```
>>> for letras in 'E-Jovem':
...     if letras == 'v':
...         pass
...         print 'Estamos só passando'
...     print 'caractere atual: ', letras
...
caractere atual: E
caractere atual: -
caractere atual: J
caractere atual: o
Estamos só passando
caractere atual: v
caractere atual: e
caractere atual: m
```

O código anterior não executa qualquer instrução ou código, se o valor da letra é 'v'. A declaração *pass* é útil quando você tiver criado um bloco de código, mas ele não é mais necessário.

Você pode então remover as instruções de dentro do bloco, mas deixar o bloco de código permanecer com uma declaração *pass* de modo que ele não irá interferir com outras partes do código.

## 3.20. Utilizando Arquivos

---

Até agora você tem utilizado a leitura e escrita para o padrão de entrada e saída do computador, que são respectivamente, o teclado e o monitor. Agora vamos ver como podemos jogar os dados em arquivos “reais” no computador.

O Python fornece funções básicas necessárias para manipular arquivos por padrão, isto é, não precisamos importar nenhum módulo. Você pode fazer sua manipulação de arquivos utilizando uma sintaxe simples que veremos a seguir.

### 3.20.1. A função *open()*

---

Antes que você possa ler ou escrever em um arquivo, você tem que abri-lo usando a função Python *open()*. Essa função cria um arquivo ou abre um arquivo que já existia. Esse arquivo a partir deste ponto poderá utilizar funções que faça tratamento de arquivos.

Python abre/cria um arquivo rapidamente para ser utilizado, veja sua sintaxe a seguir:

```
arquivo = open('caminho_do_arquivo ou nome', ['modo de acesso'])
```

Comentando sobre os parâmetros:

- caminho do arquivo: é uma *String* que contém o caminho/nome de onde o arquivo se encontra.
- modo de acesso: determina em que modo o arquivo deve ser aberto, por exemplo: leitura, escrever no final do arquivo, etc. Este parâmetro é opcional e o seu padrão é somente leitura.

Segue uma tabela com as opções mais utilizadas:

r	Abre um arquivo somente para leitura. O “cursor” do arquivo é colocado no início do arquivo.
r+	Abre um arquivo para leitura e escrita. Como no modo (r) o “cursor” estará no início do arquivo.
w	Abre um arquivo somente para escrita. Este modo sobrescreve o arquivo se ele existir, se o arquivo não existir, este é criado.
w+	Abre um arquivo para leitura e escrita. Como no modo (w) o “cursor” estará no início do arquivo.
a	Abre um arquivo para escrever conteúdo. Neste modo, o “cursor” do arquivo se encontra <b>no</b> final deste. Se o arquivo não existir ele cria um novo para escrita.
a+	Abre o arquivo, tanto para escrita quanto para leitura. O “cursor” do arquivo se encontra <b>ao</b> final deste.

Com o arquivo criado, podemos ainda tirar algumas informações dele, tais como: em qual modo o arquivo foi aberto, qual o caminho do arquivo e se o arquivo está aberto ou fechado. Seu uso é simples:

- `arquivo.closed`: Retorna verdadeiro se o arquivo está fechado;
- `arquivo.mode`: Retorna o modo de acesso com o qual o arquivo foi aberto;
- `arquivo.name`: Retorna o nome do arquivo.

Estas informações servem para fazer testes que mostram o estado do arquivo e evitar que o programa sofra alguns erros, que geralmente acontecem.

### 3.20.2. A função `close()`

Utilizamos a função `close()` quando queremos fechar o arquivo. Esta função esvazia qualquer dado que ainda não foi gravado e fecha o arquivo. Após usar esta função você não estará apto a ler ou escrever no arquivo, até que o abra novamente.

Sua sintaxe:

```
arquivo.close()
```

Podemos agora mostrar as duas funções (`open()` e `close()`) em um exemplo básico. Siga os seguintes passos:

```
coordenador@cord06:~$ mkdir logica_de_programacao
coordenador@cord06:~$ cd logica_de_programacao/
coordenador@cord06:~/logica_de_programacao$ ls
coordenador@cord06:~/logica_de_programacao$ python
...
>>> arquivo = open('nomes.txt', 'w')
>>> print 'O nome do arquivo é: ', arquivo.name
O nome do arquivo é: nomes.txt
>>> arquivo.close()
>>> arquivo.closed
True
>>> exit()
coordenador@cord06:~/logica_de_programacao$ ls
nomes.txt
```

### 3.21. Escrevendo nos arquivos com a função `write()`

---

A função `write()` escreve qualquer sequência de caracteres para um arquivo que foi previamente aberto. É importante notar que não podemos gravar tipo de dados inteiro, somente strings e binário.

A função `write()` não adiciona um caractere de nova linha (`'\n'`) ao final da sequência, veja sua sintaxe:

```
arquivo.write(string)
```

Aqui, o parâmetro “*string*” é o conteúdo que será escrito no arquivo aberto. Seguindo o exemplo anterior com o arquivo `nomes.txt`.

```
coordenador@cord06:~/logica_de_programacao$ python
>>> arquivo = open('nomes.txt', 'w+')
>>> arquivo.write("Mateus\nJosé\nJoão")
>>> arquivo.close()
>>> exit()
coordenador@cord06:~/logica_de_programacao$ cat nomes.txt
Mateus
José
João
coordenador@cord06:~/logica_de_programacao$
```

O exemplo acima abre o já criado arquivo “`nomes.txt`”, escreve o conteúdo da *string* que está sendo passado como parâmetro da função `write()`, logo em seguida o arquivo é fechado e saímos do *prompt* do Python. Já no shell do linux visualizamos o conteúdo do arquivo editado com o comando `cat` e nos é mostrado no terminal.

### 3.22. Lendo informações do arquivo com a função `open()`

---

O método `read()` lê uma sequência de caracteres (*String*) de um arquivo previamente aberto. Sua sintaxe segue a simplicidade:

```
arquivo.read([numero])
```

Aqui o parâmetro passado é opcional, este parâmetro refere-se a quantidade de *bytes* a serem lidos do arquivo. Esta função começa a ler o arquivo de seu início e se o parâmetro foi omitido, ele tenta ler o tanto que conseguir, se possível até o final do arquivo.

Retornando ao exemplo usado nas seções acima vamos novamente abrir o arquivo `nomes.txt` utilizando Python.

```
coordenador@cord06:~/logica_de_programacao$ python
>>> arquivo = open('nomes.txt', 'r') # perceba que abrimos o arquivo
somentemente para leitura
>>> nomes = arquivo.read()
>>> print nomes
Mateus
José
João
>>> exit()
coordenador@cord06:~/logica_de_programacao$
```

Fazendo uso do parâmetro opcional podemos imprimir apenas até um exato número de *bytes*, neste caso, uma letra equivale a 1 *byte*. Se quisermos ler o primeiro nome completo, contamos quantos caracteres tem esta palavra, é este o valor que deve ser posto como parâmetro da função.

```
coordenador@cord06:~/logica_de_programacao$ python
>>> arquivo = open('nomes.txt', 'r') # perceba que abrimos o arquivo
somentemente para leitura
>>> nome1 = arquivo.read(6)
>>> print nome1
Mateus
>>> exit()
coordenador@cord06:~/logica_de_programacao$
```

### 3.22.1. Posição do cursor

A função `tell()` te informa a posição atual do cursor no arquivo aberto, em outras palavras, a próxima leitura ou gravação ocorrerá a partir do *byte* informado por esta função.

```
coordenador@cord06:~/logica_de_programacao$ python
>>> arquivo = open('nomes.txt', 'a+') # perceba que abrimos o arquivo para
adicionar conteúdo ou para leitura
>>> arquivo.tell()
0L
>>> arquivo.close()
>>> exit()
coordenador@cord06:~/logica_de_programacao$
```

Uma outra função que nos ajuda a nos locomover dentro do arquivo é a função `seek()`. Ela nos permite mudar a posição do “cursor” no arquivo aberto atualmente. Segue sua sintaxe:

```
arquivo.seek(offset [, from])
```

O parâmetro `offset`, indica quantos bytes serão movidos. Já o parâmetro `from` (opcional) aceita três valores possíveis, listados abaixo:

- `from = 0`: Significa que a função `seek()` vai utilizar o início do arquivo como referência para se locomover internamente;

- `from = 1`: Neste caso a função `seek()` vai utilizar a posição atual como referência da movimentação;
- `from = 2`: O final do arquivo é tomado como referência.

Temos o seguinte exemplo:

```
coordenador@cord06:~/logica_de_programacao$ python
>>> arquivo = open('nomes.txt', 'a+')
>>> f.seek(7) # nesta linha de código pulamos o primeiro nome: Mateus
>>> f.read()
José
João
>>> f.tell()
19L
>>> f.seek(7,0) # novamente pulamos o primeiro nome: Mateus
>>> f.seek(6,1) # pulamos agora o segundo nome: José
>>> f.read()
João
>>> exit()
coordenador@cord06:~/logica_de_programacao$
```

**Exercício:** Crie arquivos com algumas listas de nomes com as funções aprendidas acima.

# Projeto -Jovem

**Cid Ferreira Gomes**

*Governador do Estado do Ceará*

**Maria Izolda Cela de Arruda Coelho**

*Secretária da Educação do Estado*

**Maurício Holanda Maia**

*Secretário adjunto da Secretaria da Educação do Estado*

**Professor Cláudio Ricardo Gomes de Lima Msc.**

*Reitor do Instituto Federal de Educação, Ciência e Tecnologia do Ceará*

**Professor Edson da Silva Almeida Msc.**

*Diretor Executivo do CPQT*

**Andrea Araújo Rocha**

*Coordenadora Geral do Projeto e-Jovem – SEDUC*

**Professor Cícero R. B. Calou Msc.**

*Gerente de Projetos do CPQT*

*Coordenador do Projeto e-Jovem - Módulo II - IFCE-CPQT*

**Júlio César Cavalcante Bezerra**

*Coordenador do Projeto e-Jovem - Módulo II - SEDUC*

**Vitor de Carvalho Melo Lopes**

*Projeto e-Jovem - Módulo II*

*Edição de Conteúdo*

**Jucimar de Souza Lima Junior**

*Projeto e-Jovem - Módulo II*

*Edição de Conteúdo*

**Francisca Lúcia Sousa de Aguiar**

*Projeto e-Jovem - Módulo II*

*Revisão*

**Martha Aurélio Moreira de Melo**

*Projeto e-Jovem - Módulo II*

*Revisão*

**Rafael de Sousa Lima**

*Projeto e-Jovem - Módulo II*

*Formatação Final*