Fundamentos da Computação com Python

Prof. Alexandre Rojas
Departamento de Informática e
Ciência da Computação IME/UERJ

29 de agosto de 2017

Sumário

Prefácio

Voltado para alunos iniciantes e intermediários no curso de desenvolvimento de algoritmos e na linguagem Python o livro visa o aprendizado da programação estruturada e sua aplicação em uma linguagem de fácil aprendizado. Durante o decorrer do livro serão apresentados diversos conceitos úteis não somente para alunos de ciência da computação, mas também para alunos de engenharia, ou qualquer outro aluno que queira iniciar o aprendizado em programação.

Como o leitor já deve ter percebido, ao contrário dos demais livros, que são extremamente longos este livro tentará ser o mais breve possível, para que dessa forma possa auxiliar estudantes em sala de aula ou em casa.

Capítulo 1

Computadores e algoritmos

A construção de programas é a questão central para o uso eficaz dos computadores na solução de problemas, a partir do desenvolvimento de conjuntos de funções organizadas de maneira lógica. As regras para um raciocínio correto já eram conhecidas dos antigos gregos no século IV A.C. com os ensinamentos de Sócrates, Platão e Aristóteles a quem coube criar o grupo conhecido como Lógica Formal, Lógica menor ou Epistemologia que contém a sistematização das leis.

A interpretação matemática dessas regras lógica somente ocorreu em 1854, com George Boole em seu trabalho "Uma investigação das leis do pensamento". Este conjunto de conhecimentos é hoje chamado de Álgebra Booleana compondo-se de Lógica Simbólica e Lógica Matemática. Quase um século mais tarde, em 1937, Claude Elwood Shannon publicou o trabalho "Uma análise simbólica de circuitos de comutação e relés" mostrando que a Lógica Simbólica de Boole descrevia os circuitos lógicos como a base para o projeto dos computadores.

1.1 A obtenção da Informação

A informação tem sua origem na realidade que nos cerca. A partir desta realidade pode-se realizar uma abstração gerando o conhecimento.

O conhecimento de um objeto existente no mundo real é constituído por um conceito que reflete seu tipo e um termo. O termo é o que nos permite comunicar a ideia e processamento de dados é denominado valor. Um termo ou valor pode ter várias representações, conforme mostra a figura 1 a seguir:

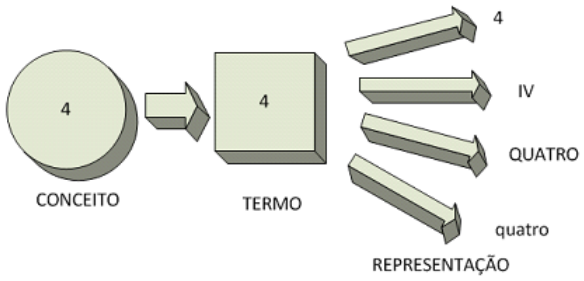


Figura 1.1: Representações

A informação, conceito ou termo existe no mundo lógico, enquanto o que denominamos dados existe no mundo físico. A partir da representação a informação poderá ser obtida por uma operação de interpretação.

Pode-se perceber que existem dois grandes grupos de informações: o primeiro trata de informação estruturada como, por exemplo, os dados pessoais de um grupo de pessoas ou mesmo uma pesquisa no Google; o segundo grupo trata de informações não estruturadas como, por exemplo, a Web Semântica que trata da integração de recursos computacionais e humanos para ajudar na descoberta e no uso dos recursos da web.

Neste livro iremos utilizar informações estruturadas. Inicialmente deve-se distinguir a informação inicial da informação estruturada.

Para que uma informação seja processada é necessário um processo prévio de entendimento e modelagem do problema cuja sequência de operações pode ser observada na figura

Figura 1.2: Etapas da Modelagem das informações

1.2 Programas Tradutores

A partir do momento que conseguimos a informação estruturada, podemos transcrever essa informação para o computador. Porém, o computador só entende o que chamamos de linguagem de máquina, escrito em numeração binária. A princípio, o computador só entende 0 ou 1.

Para facilitar a criação de programas de computadores foram criados os programas tradutores, que tem a finalidade de transformar os códigos escritos pelos programadores, em um código que o computador entenda. Mais especificamente, um tradutor transforma um código em uma linguagem de mais alto nível em uma linguagem de mais baixo nível. A seguir são descritos os dois principais tipos de tradutores: interpretadores e compiladores.

1.2.1 Interpretadores

Os interpretadores leem um código fonte e convertem para um código executável.

Vantagens: Depuração do programa é mais simples, consome menos memória, resultado imediato na rotina desenvolvida.

Desvantagens: Execução do programa é mais lenta, estruturas de dados são simples, necessário fornecer o programa fonte ao utilizador.

1.2.2 Compiladores

É um tipo de tradutor que converte o código fonte reconhecido pela linguagem para a linguagem de máquina, aquela que o computador pode entender.

Vantagens: Execução rápida, permite estruturas de programação mais completas, permite otimização do código fonte

Desvantagens: Várias etapas de tradução, maior consumo de memória, processo de correção de erros e depuração é mais demorado.

1.2.3 Erros

Os erros em Python podem léxicos ou sintáticos.

- Erros Léxicos: Os erros léxicos caracterizam-se por uma escrita errada ou simplesmente não reconhecida pela linguagem adotada. O alfabeto utilizado para escrever os comandos é composto pela tabela ASCII (American Standard Code for Information Interchange). O Python a partir da versão 2.x passou a suportar os caracteres do UNICODE. O UNICODE é uma evolução do padrão ASCII que permite aos computadores representar e manipular, de forma consistente, os caracteres de qualquer sistema de escrita existente.

```
>>> a="Σ"
Unsupported characters in input
```

- Erros sintáticos: Os erros sintáticos caracterizam-se pela a utilização errada ou desconhecida de um ou mais conjuntos de vocábulos.

```
>>> 3+5 #modo correto de declarar uma soma
8
>>> três mais cinco
SyntaxError: invalid syntax
>>> tres mais cinco
SyntaxError: invalid syntax
>>> three plus five
SyntaxError: invalid syntax
```

- Erros semânticos: Refere-se a operações que fazem sentido para o tradutor, mas não para o usuário.

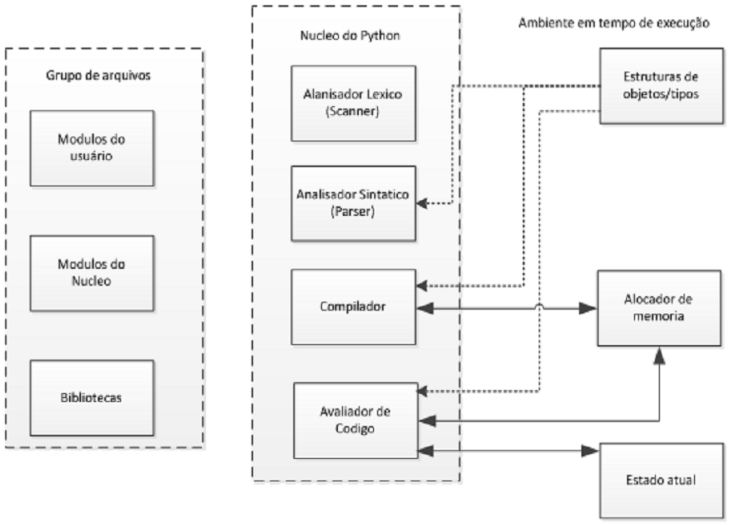
```
>>> 3+2*5 # (3+2)*5
13
```

1.3 Funcionamento da linguagem

Em resumo, durante o processo de compilação e execução do código, que ocorre quase que de forma oculta ao programador, podemos citar algumas etapas chave:

- Compilação do código fonte em código de byte: Nessa etapa o código fonte, conteúdo do programa, é transformado em um código em linguagem de máquina e alocado em um local seguro no seu computador. Esses arquivos permanecerão salvos a fim de agilizar o processo de compilação (o próprio Python verifica se as datas de alteração do código fonte e do código de bytes para saber se deverá ou não recompilar).

- Interpretação do código de bytes: Uma vez que o código de bytes tenha sido gerado, ele poderá ser interpretado etapa a etapa. O esquema abaixo mostra de forma resumida os processos realizados pela linguagem:



O diagrama acima foi elaborado por Bastos, Eduardo e Freitas, Juliano no documento "Interpretador / Compilador Python" e apresenta a arquitetura do Python que pode ser classificada em quatro componentes principais. O diagrama acima representa a interpretação da arquitetura do nível mais elevado do sistema Python.

No lado esquerdo, são agrupados as bibliotecas, módulos do núcleo e módulos definidos pelos usuários. A biblioteca e os módulos definidos pelos usuários podem estender o sistema Python desde que seja mantida a portabilidade. No centro do diagrama esta ilustrado o núcleo do interpretador Python. As setas dentro da caixa do interpretador indicam o fluxo de dados.

No lado direito são mostrados o estado atual do Python, o alocador de memória e a estrutura de objetos e tipos, que constituem o ambiente em tempo de execução. O estado atual do Python refere-se ao estado de execução do interpretador, visto como uma máquina de estado finito muito grande, complexa, e modificável. O alocador de memória é responsável por alocar a memória para objetos de Python (externo e interno) e conectado com as rotinas padrão malloc do C. As estruturas de objetos e tipos representam os objetos internos que estão disponíveis em Python

1.4 Sobre o Python

A linguagem Python foi concebida no final de 1989, por Guido van Rossum, contudo seu código só foi publicado em 1991, influenciada pela linguagem ABC, que visava o aprendizado

e competia com as linguagens Basic e AWK, possuindo a vantagem de escrever bem menos em comparação a essas linguagens. Assim, surgiu a ideia da criação de uma linguagem que englobasse os elementos bem sucedidos na linguagem ABC, que eliminasse a declaração de variáveis e que usasse a indentação para delimitar seus blocos baseando-se no ALGOL.

O Python é uma **linguagem interpretada** licenciada como Software Livre e com uma vasta biblioteca de módulos igualmente livres e sem restrições para sua distribuição. A natureza flexível da linguagem de programação Python suporta diversas filosofias de programação, incluindo programação procedural, orientada a objetos e funcional.

1.5 Características da linguagem

- Conceitos fundamentais fáceis de compreender
- Programas são mais curtos de escrever
- Sintaxe clara e concisa, favorecendo o entendimento.
- Presença de estruturas de alto nível.
- Diversos módulos já inclusos.
- Possibilidade de download de bibliotecas de terceiros.
- Suporte a programação modular, funcional e à orientação a objeto.
- Para Python "tudo é objeto".
- Apesar de ser considerada linguagem interpretada, em Python existe a possibilidade de compilar seu código.
- Python é livre.
- Pode ser executado em qualquer plataforma.
- Python é case sensitive, ou seja, ele faz diferenciação entre letras maiúsculas e minúsculas.
- Não é tão eficiente como algumas linguagens compiladas.
- Possibilidade de integrar Python com outras linguagens.
- Possui tipagem dinâmica, ou seja, os tipos das variáveis são definidos pelo interpretador, sem a necessidade do programador definir seu tipo previamente.
- A tipagem é forte, isso significa que a linguagem não converte tipos automaticamente, como ocorre em algumas linguagens, por esse motivo ao realizar as operações deve ser claro o tipo da variável que está usando.

1.6 Quem usa Python

Nasa, Google, Universidade de Maryland, Embratel, CPqD, Conectiva, Async, Sempro, Apple (MAC OS X), Microsoft (.NET) , Disney, Nokia, Atari, Yahoo!, Philips, Blender 3D, PostgreSQL, Inkscape entre outros.

1.7 Instalação e inicialização do Python no Windows

Faça o download do arquivo de instalação do Python na página da Internet: http://www.Python.org/downlo selecione a opção para download do Python 2.7.10

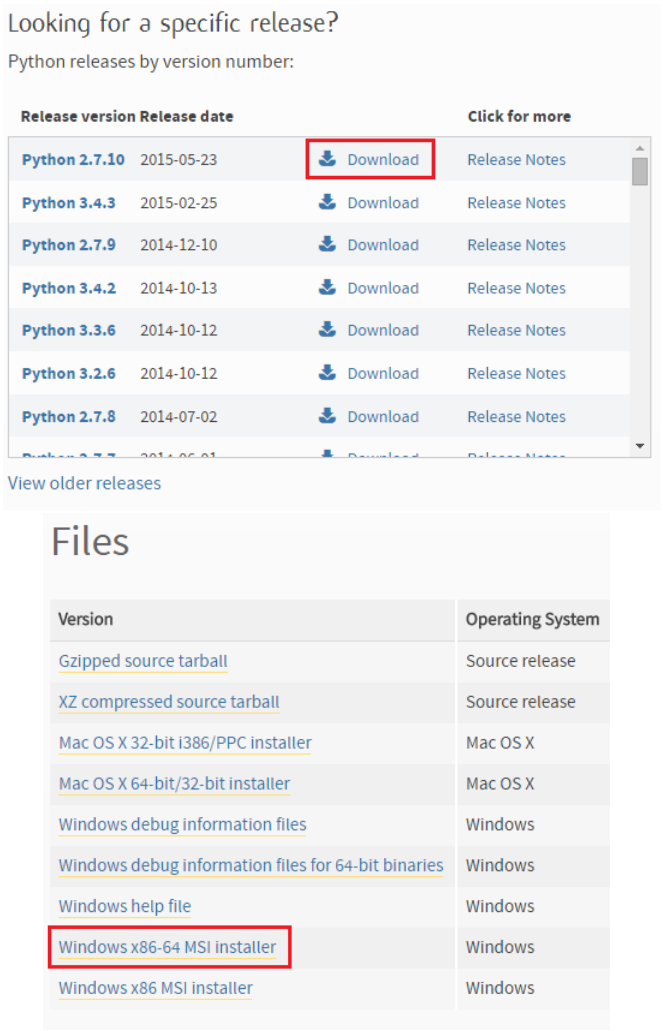


Figura 1.3: Telas de download do Python, onde foi selecionado a versão 2.7.10 e, em seguida, o Windows x86-64

Execute este arquivo dando um duplo clique com o botão esquerdo do mouse sobre ele e proceda com a instalação normalmente. Ao final do processo terão sido instalados os seguintes componentes em seu computador:

IDLE (Python GUI) – Ambiente de Desenvolvimento Integrado disponibilizado junto com o Python e que será utilizado neste livro;

- Module Docs – Documentação dos Módulos do Python;
- Python (Command Line)- Interpretador usando Linhas de Comando
- Python Manuals – Manuais do Python em Inglês;
- Uninstall Python – Desinstalador da Linguagem

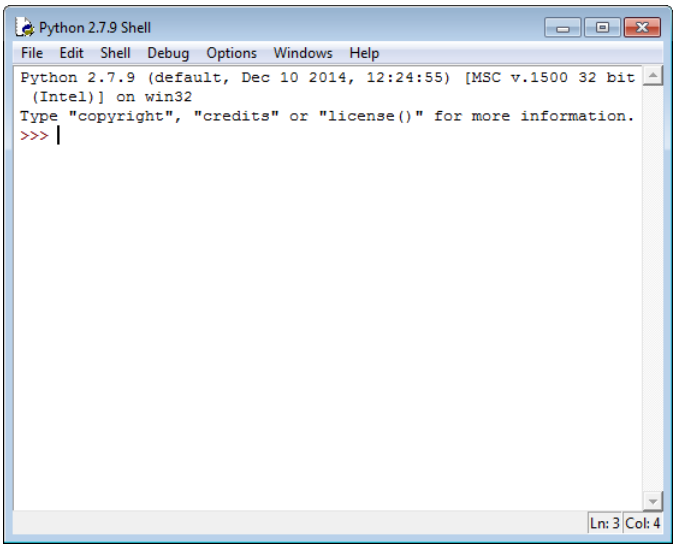
1.8 Modo Interativo (IDLE)

Embora possamos selecionar várias interfaces gráficas (GUI) para usarmos com o Python neste livro usaremos a IDLE por ser de fácil uso e aprendizado.

Para usá-la siga os seguintes passos:

a) Selecione

b) A tela do Shell do Python para processamento interativo é mostrada a seguir:



Neste módulo os comandos digitados seguidos de [ENTER] serão executados imediatamente possibilitando que o Python possa ser usado como se fosse uma calculadora.

1.8.0.1 Alguns exemplos no Modo Interativo

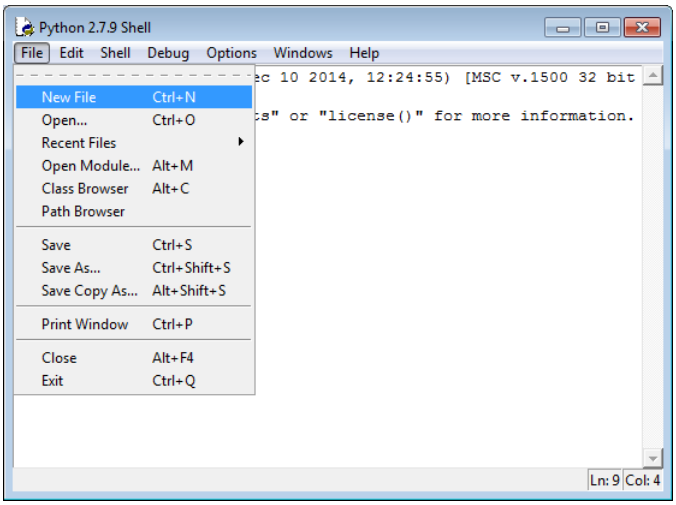
```
>>> 2+2
4
```

Lembre-se que o Python usa o ponto para separar a parte inteira de um numero da parte fracionária, seguindo o padrão americano de formatação numérica.

```
>>> 10.5/3
3.5
>>> 10,5/3
(10, 1)
```

1.9 Modo Programado

Para usar o Python no modo programado, inicie uma nova janela.



Neste modo os comandos serão

- Digitados;
- Salvos – Atenção Não se esqueça de colocar a extensão py no seu arquivo. O Python NÃO faz isto automaticamente;
- Executar o programa pressionando F5 ou selecionando

Apesar de não ser exatamente necessário, uma vez que o Python possui seu próprio editor de texto, pode-se utilizar qualquer editor de texto puro para escrever o código, como por exemplo o **Bloco de Notas** do Windows ou o **vi** do Linux.

1.10 Ajuda (Help Modules)

O Python possui diversos módulos de ajuda que podem ser acessados, contendo uma breve descrição e comandos relacionados aquilo que foi pedido.

Para acessar um help module deve-se digitar na IDLE help (argumento).

```
>>> help(str)
```

Capítulo 2

Algoritmos

Um algoritmo é uma sequência ordenada e finita de passos, que seguem regras bem definidas, e tem como finalidade a solução de um problema em um espaço de tempo finito.

Ainda que o termo tenha sido popularizado com a computação, algoritmos podem se referir a solução de qualquer tipo de tarefa. Dessa forma, por exemplo, é correto falar algoritmo de preparação de um bolo, quando quer se referir a uma receita.

2.1 Características de um Algoritmo

- Integridade: refere-se à precisão das informações manipuladas pelo programa;
- Clareza: refere-se à facilidade de leitura do programa;
- Simplicidade: a clareza e precisão de um programa são normalmente melhoradas tornando seu entendimento o mais simples possível, consistente com os objetivos do programa;
- Eficiência: refere-se à velocidade de processamento e a correta utilização da memória;
- Modularidade: consiste na divisão do programa em módulos menores bem identificáveis e com funções específicas;
- Generalidade: é interessante que um programa seja tão genérico quanto possível de forma a permitir a reutilização de seus componentes em outros projetos.

2.2 Formas de Descrever um Algoritmo

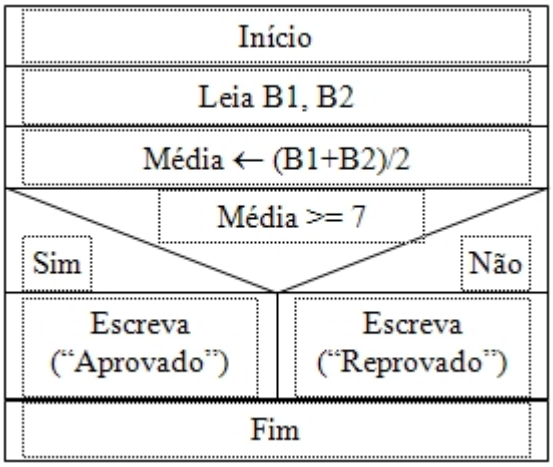
2.2.1 Descrição Narrativa

Consiste em descrever, de forma narrativa cada um dos passos necessários para a resolução de um problema.

Exemplo: Manuais, Receitas de bolo em livros de culinária, entre outros.

2.2.2 Diagrama de Chapin

Esse tipo de representação de um algoritmo visa à estruturação hierárquica da lógica de solução de um problema. A vantagem na utilização desse tipo de representação é a vantagem na localização dos pontos de divisão. Observe o exemplo abaixo que calcula a média aritmética de duas notas e verifica se o aluno foi aprovado ou não:



2.2.3 Fluxograma

É um tipo de diagrama que normalmente é utilizado para descrever um fluxo de processos. O objetivo de um fluxograma é mostrar a transição de informações entre os elementos que o compõe.

Para descrever algoritmos, os fluxogramas são compostos basicamente de elipses, retângulos e losangos. As elipses são usadas para representar o início e o fim dos algoritmos. Já, os retângulos são usados para representar o processamento de dados, como os cálculos e atribuições. Por sua vez, os losangos são usados para decisão (ou desvio), quando for necessário mudar o fluxo de informações. Além disso, as setas são usadas para representar a passagem de informações entre as figuras.

O fluxograma a seguir representa a leitura de um número inteiro e a impressão do módulo desse número.

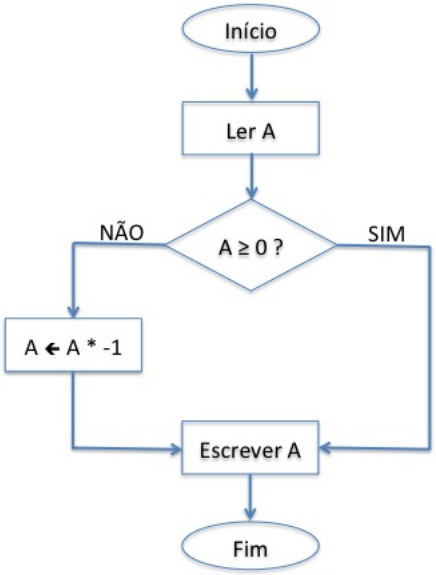
2.2.4 Pseudocódigo

Por ser a maneira mais simples e prática de representar um algoritmo, o pseudocódigo será a forma de representar um algoritmo utilizada neste livro. Um pseudocódigo simula a solução do problema através de uma linguagem computacional fictícia, mas muito próxima de uma linguagem real.

Apesar de não existirem regras rígidas para a codificação dos algoritmos através do pseudocódigo, em geral um padrão de código é seguido.

Observe a seguir um exemplo de um pseudocódigo:

```
Inicio
    Ler A,B
```




```
    C=(A+B)/2
    Se C >= 7 então
        Escrever aprovado
    Se não
        Escrever reprovado
    fim se
Fim
```

A próxima seção apresenta a notação de pseudocódigo que será adotada neste livro.

2.3 Algoritmos com Pseudocódigo

A seguir serão descritos os principais conceitos para o desenvolvimento de algoritmos utilizando o pseudocódigo.

2.3.1 Tipos de Dados

Os algoritmos recebem dados como entradas, manipulam esses dados e produzem resultados. Logo, uma parte crucial dos algoritmos é a representação dos dados do programa. Esses dados podem ser as vendas de uma loja, o nome de um funcionário, as notas dos alunos entre muitos outros.

Perceba que os dados podem ser classificados entre diferentes tipos. Por exemplo, as vendas de uma loja são números reais, enquanto que o nome de um funcionário é uma cadeia de caracteres. Os tipos de dados mais utilizados são numéricos, literais e lógicos.

Neste livro, utilizaremos os tipos de dados **Inteiro**, **Real**, **Lógico**, **Char** e **Cadeia**, para

representar números inteiros, números reais, valores lógicos verdadeiro ou falso, caracteres de tamanho um e cadeias de caracteres com qualquer tamanho, respectivamente.

2.3.2 O Conceito de Variável

Na computação, uma variável é uma entidade que possui um valor, sendo referenciada no programa por um nome. Similar ao conceito de variáveis da matemática.

O nome de uma variável segue a regra de criação de identificadores da linguagem. Um identificador é qualquer nome permitido pela linguagem. Logo, uma variável, assim como as palavras reservadas, são identificadores.

Neste livro, a regra de criação de identificadores adotada é a seguinte:

- Os caracteres permitidos são números, letras maiúsculas ou minúsculas e o caractere sublinhado;
- O primeiro caractere obrigatoriamente deve ser uma letra ou o caractere sublinhado.

Desse modo, a seguir são apresentados alguns identificadores válidos e outros inválidos.

Válidos: A, a, nota, N1, _1, A111, X5, Nota_1, dia, idade, matrícula.

Inválidos:

5A	Não pode começar com número
nota final	Não é permitido o espaço em branco
x-y	Não é permitido operadores
km/h	Não é permitido operadores
AB;	Não é permitido caractere especial
média	Não é permitido caractere especial

2.3.3 Declarando uma Variável

Toda variável a ser utilizada no algoritmo deve ser declarada na área especificada. Para declarar uma variável, basta colocar o tipo da variável seguido do nome da variável ou de uma lista de nomes separados por vírgula.

Ex: Inteiro idade
Real N1, N2
Cadeia nomeAluno
Char sexo
Lógico resultado

2.3.4 Atribuição(←)

O comando ← é lido como atribua. Uma operação a ← b+c, significa "pegue conteúdo de b e aplique a operação soma com o conteúdo de c, o resultado dessa operação armazene em a".

Ex: N1 ← 6.0
sexo ← 'M'
resultado ← verdadeiro
idade ← idade + 1

2.3.5 Operadores Aritméticos

+	soma
-	subtração
*	multiplicação
/	divisão real
//	divisão inteira
%	resto inteiro da divisão

Ex: 5/2 - o resultado é 2.5
5//2 - o resultado é 2
5%2 - o resultado é 1

2.3.6 Operadores Relacionais

São usados para comparar valores, sendo o resultado da operação sempre um valor lógico verdadeiro ou falso. Podemos usar os seguintes operadores relacionais:

>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
!=	diferente
==	igual

Ex: (N1+ N2)/2 >= 7.5

2.3.7 Operadores Lógicos

São usados quando queremos unir duas ou mais expressões lógicas, em geral, comparações.

Os principais operadores lógicos são: e, ou, não

O resultado das operações lógicas pode ser compreendido através da tabela verdade, conforme é mostrado a seguir:

C1	C2	C1 e C2	C1 ou C2	Não C1
V	V	V	V	F
F	V	F	V	V
V	F	F	V	F
F	F	F	F	V

Nessa tabela C1 e C2 são condições a serem testadas, V equivale a verdadeiro e F a falso.

2.3.8 Comandos de Entrada/Saída

Os cálculos do computador são de pouco valor se não podemos fornecer os dados sobre os quais os cálculos serão efetuados e se não podemos ver os resultados destes cálculos.

Uma Entrada significa receber um valor do usuário do programa que é armazenado em alguma variável. Por exemplo, o usuário informar as notas de um aluno.

Uma Saída significa mostrar o resultado de um cálculo ou de uma variável para o usuário do programa.

Escrever

Significa exibir o conteúdo de uma variável. Não destrói o valor exibido. Pode-se exibir uma mensagem colocando entre aspas.

Ex: Escrever num1
    Escrever "nota:", nota

Ler

Significa receber um conteúdo digitado pelo usuário do programa e armazená-lo em uma variável.

Ex: Ler nota
    Ler nome, sexo, salario

2.3.9 Estrutura de Decisão

As estruturas de decisão (ou também conhecidas como desvios) são usadas quando queremos mudar o fluxo do programa, ou seja, quando queremos que uma ou mais instruções sejam executas ou não dependendo de uma condição.

- Decisão simples: Quando existe apenas uma ação a ser executada se a condição for verdadeira;
- Decisão composta: Quando existem duas ações a serem executadas se a condição for verdadeira ou se for falsa;

2.3.9.1 Representação de uma Estrutura de Decisão

Simples

```
Se condição então
    Passos a serem executados
Fim se
```

Os passos só serão executados se a condição for verdadeira.

Composta

```
Se condição então
    Passos1 a serem executados
```

```
Senão
    Passos2 a serem executados
Fim se
```

Os passos1 só serão executados se a condição for verdadeira. Por outro lado, os passos2 só serão executados se a condição for falsa.

2.3.10 Estrutura de Repetição

As estruturas de repetição são usadas quando queremos que um grupo de comandos seja executado mais de uma vez. As estruturas de repetição podem ser determinadas, quando sabemos a quantidades de vezes queremos repetir o código (estrutura para), e indeterminadas, quando sabemos apenas que o código deve repetir até atingir uma determinada condição (estrutura enquanto).

Representação da Estrutura Para Faça

```
Para variável = início até fim
    Passos a serem executados
Fim para
```

Funcionamento da Estrutura Para Faça

1. A variável é inicializada com o valor do início;

2. O grupo de instruções é executado;

3. Ao chegar ao comando "fim para" a variável tem seu valor acrescido de 1(um) e verifica se o seu valor é maior que fim;

4. Se for menor ou igual, volta para a instrução 2, senão termina a repetição.

Representação da Estrutura Enquanto Faça

```
Enquanto condição faça
    Passos a serem executados
Fim enquanto
```

Funcionamento da Estrutura Enquanto Faça

Enquanto a condição for verdadeira o grupo de instruções dentro do enquanto é repetido. No momento que a condição ser tornar falsa a repetição termina. Desse modo, se inicialmente a condição for falsa o grupo de instruções não será executado nenhuma vez. Por outro lado, caso a condição nunca se torne falsa a repetição NUNCA irá terminar, condição que é denominada "entrar em loop".

2.4 Exercícios Resolvidos

1. Escreva um algoritmo para determinar a soma de 2 valores lidos.

```
Algoritmo Soma
    Inteiro A, B, C
Inicio
    Ler A
    Ler B
    C ← A + B
    Escrever "Soma:", C
Fim
```

2. Escreva um algoritmo que leia dois valores e imprima seu produto.

```
Algoritmo Produto
    Inteiro A, B
Inicio
    Ler A
    Ler B
    Escrever "Produto:", A * B
Fim
```

3. Escreva um algoritmo que imprima a tabuada de 1 a 10 de um número a ser lido.

```
Algoritmo Tabuada
    Inteiro I, A, Resultado
Inicio
    Ler A
    Para I=1 a 10 faça
        Resultado ← A*I
        Escrever A, "*", I, "=", Resultado
    Fim para
Fim
```

4. Escreva um algoritmo que calcule o fatorial de um número qualquer.

```
Algoritmo Fatorial
    Inteiro N, I, FAT
Inicio
    Ler N
    FAT ← 1
    Para I=1 até N faça
        FAT ← FAT*i
    Fim para
    Escrever "Fatorial:", FAT
Fim
```

5. O termo "swap" indica a troca de valor entre duas variáveis. Escreva um algoritmo para a técnica swap entre duas variáveis.

```
Algoritmo Troca
    Inteiro A, B, AUX
Inicio
    Ler A, B
    Escrever A, B
    AUX ← A
    A ← B
    B ← AUX
    Escrever A,B
Fim
```

6. Escreva um algoritmo que gere a seguinte saída:
   1: Domingo 2: segunda-feira 3: terça-feira 4: quarta-feira 5:quinta-feira 6:sexta-feira 7: sábado

```
Algoritmo Semana
    Inteiro i
Inicio
    i ← 1
    Escrever i + ": domingo"
    i ← i + 1
    Escrever i + ": segunda-feira"
    i ← i + 1
    Escrever i + ":terça-feira"
    i ← i + 1
    Escrever i + ":quarta-feira"
    i ← i + 1
    Escrever i+ "quinta-feira"
    i ← i + 1
    Escrever i + " sexta feita"
    i ← i + 1
    Escrever i + "sábado"
Fim
```

7. Escreva um algoritmo para ler um número e imprimir se ele é igual a 5, 200 ou 400. Se não, verificar se o mesmo está no intervalo entre 500 e 1000 inclusive, ou se está fora do escopo especificado.

```
Algoritmo Intervalo
    Inteiro a
Inicio
    Ler a
    Se (a==5) ou (a==200) ou (a==400) então
        Escrever "igual a " , a
    Senão
        Se (a>=500) e (a<=1000)
            Escrever "dentro do intervalo verificado"
        Senão
```

```
                Escrever "Fora do escopo"
            Fim se
        Fim se
    Fim
```

8. Escreva um algoritmo que leia três valores para A, B e C respectivamente. O programa devera imprimir qual das três letras tem o maior valor.

```
Algoritmo MAIOR
    Inteiro A,B,C
Inicio
    Ler A,B,C
    Se A>=B então
        Se A>C então
            Se A==B então
                Escrever "A e B"
            Senão
                Escrever "A"
            Fim se
        Senão
            Se A==C então
                Se A==B então
                    Escrever "A, B e C"
                Senão
                    Escrever "A e C"
                Fim se
            Senão
                Escrever "C"
            Fim se
    Senão
        Se B>C então
            Escrever "B
        Senão
            Se B==C então
                Escrever "B e C"
            Senão
                Escrever "C"
            Fim se
        Fim se
    Fim se
Fim
```

9. O sistema de avaliação de uma determinada matéria é dada da seguinte maneira:
prova1 + prova2 >= 7 → aluno é aprovado
prova1 + prova2 < 4 → aluno é reprovado
prova1 + prova2 >= 4 e prova1 + prova2 < 7 → recuperação
Escreva um algoritmo que leia 5 valores de prova1 e prova2 e imprima o status do aluno.

```
Algoritmo Resultado5
    Real p1, p2
    Inteiro k
Inicio
    Para k=1 até 5 faça
        Ler p1
        Ler p2
        Se (p1+p2)/2.0 ¡ 4 então
            Escrever "aluno reprovado"
        Senão
            Se (p1+p2)/2.0>=7 então
                Escrever "aluno reprovado"
            Senão
                Escrever "prova final"
            Fim se
        Fim se
    Fim para
Fim
```

10. Qual seria o enunciado do problema que teria como resultado o seguinte algoritmo:

```
Algoritmo Nome
    Inteiro a
Inicio
    a = 0
    Enquanto a < 100 faça
        Se (a < 48) ou a >= 73 então
            a = a + 3
            Imprimir a
        Se não
            a = a + 5
            Imprimir a
        Fim se
    Fim enquanto
Fim
```

Resposta: Elabore um algoritmo que imprima todos os números de 1 a 100 múltiplos de 3 se eles forem menores que 50, de 5 em 5 para valores maiores ou iguais a 50 e menores que 75, e de 3 em 3 para valores maiores ou iguais a 75.

2.5 Exercícios Propostos

1. Supondo que a população mundial seja estimada em 90 bilhões com uma taxa de crescimento anual de 1%, escreva um algoritmo que calcule a quantidade estimada de habitantes no decorrer de 10 anos.

Capítulo 3

Primeiros Passos

Este capítulo apresenta uma introdução a linguagem Python, apresentando como declarar identificadores, as palavras reservadas, o comado de atribuição, separadores e delimitadores, entre outros aspectos iniciais.

3.1 Variáveis ou Identificadores

O conceito de variável é uma associação entre um nome e um valor. Em Python, tudo é **objeto** assim, uma variável é apenas um **nome** que faz **referência** a um **objeto**. Estes objetos podem ser **mutáveis** ou **imutáveis**.
Uma variável passa a existir no momento que se faz a atribuição de um valor para ela. O Python em uma tipagem dinâmica que significa que a natureza de uma variável pode ser alterada em tempo de processamento e seu tipo é aquele do valor atribuído. A seguir um exemplo:

```
>>> nota=9.
```

Este comando pode ser interpretado como *atribua o valor do tipo numérico real 9 a variável nota*.
Para atribuir um valor a uma variável, o Python faz 3 coisas: (a) cria um objeto do tipo float que possui o valor 9,(b) cria o nome *nota* e (c) cria um ponteiro(pointer) que faça a ligação entre o objeto do tipo float e aquele que tem valor 10. Assim, toda vez que a variável nota for usada ela será automaticamente substituída pelo valor do objeto ao qual esta variável faz referência.
Veja o diagrama a seguir:

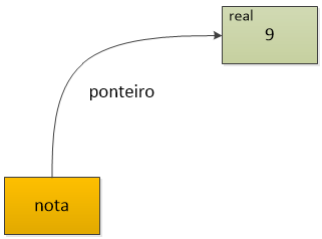


Figura 3.1: Ligação variável valor

Porem se fizermos o comando nota+=1, ou seja, adicione o valor 1 a variável nota, o Python cria um novo objeto e passa a referenciar a este.

```
>>> nota+=1.
```

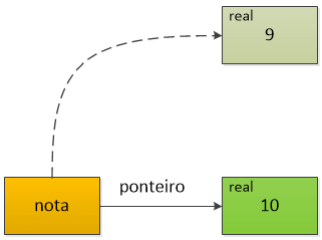


Figura 3.2: Alterando o valor da variável

O diagrama a seguir mostra a situação onde ocorre a igualdade entre duas variáveis.

```
>>> nota=media= 8.
```

Nota e média passam a apontar para o mesmo objeto.

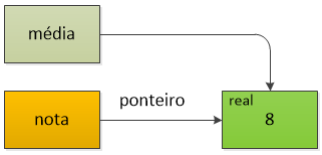


Figura 3.3: Duas variáveis iguais a um valor

Este tipo de variável é denominado **imutável**, pois o objeto não é alterado e sim criado um novo. Desta forma, o primeiro objeto com valor 9 deixa de ser referenciado e permanece na memória. Posteriormente, o *garbage collector* ,ou seja, o coletor de lixo de objetos na memória não referenciados apaga estes objetos. O Python preserva os objetos tipo *int* ou *str* que ocupam pouca memória para evitar o "esforço"de recria-los. Uma variável do tipo **mutável** é do tipo que permite alterar o objeto apontado sem a criação de um novo objeto. Um exemplo de variável **mutável** são as listas, que serão melhor detalhadas posteriormente. Para facilitar o entendimento pense em uma lista de valores separadas por virgula.
Por exemplo:x é uma lista que contem os valores 1, 2 e 3

```
>>> x=[1,2,3]
```

Visualmente temos:

Figura 3.4: Variável mutável

Pode-se alterar qualquer dos elementos de x sem que o Python crie um novo objeto.

Ao criar uma variável deve-se ficar atento às seguintes regras:

- O Python é "Case Sensitive" isto significa que letras maiúsculas são diferentes de minúsculas. Dessa forma, podemos criar diversos identificadores com a mesma letra alternando somente a caixa e todos eles serão tratados como coisas diferentes;
- Podem ter qualquer quantidade de caracteres;
- Podem ter **letras e números** ou **underscore** (sublinhado _);
- **Não** podem ter **espaços em branco** ou **caracteres especiais**, Ex: ()*&%$#!+-=/?
- **Iniciam** sempre por uma **letra**;
- **Não** podem ser uma **palavra reservada**.

3.1.1 Palavras reservadas

and	exec	not
break	for	pass
global	raise	from
while	import	try
else	i	with
assert	finally	or
class	print	continue
def	if	return
elif	in	del
except	lambda	yield

Exemplos de variáveis válidas: Idade, alfa23, ano_de_nascimento, Nome_do_ Aluno

Exemplos de variáveis NÃO válidas: Ano de nascimento, alfa*15, r&10

3.2 Comandos de Atribuição

Um comando de atribuição é sempre do tipo

variável = expressão.

O símbolo = significa **atribua** e tem sentido diferente do = (igual) da matemática (que em python é ==).

Observe o exemplo:

```
>>> a=5
>>> nome='uerj'
>>> a,b=2,3
```

3.3 Linhas

Um programa em Python é dividido em linhas lógicas que podem ser compostas de uma ou mais linhas físicas. Uma linha lógica não pode ultrapassar uma linha física com exceção de dois casos especiais:

Colocando-se \ no final da linha. Por exemplo:

```
>>> jan=1
>>> fev=2
>>> mar=3
>>> abr=4
>>> total=jan+fev+\
mar+abr
>>> total
10
```

Ou quando ocorre o ajuntamento implícito de linhas dentro de expressões delimitadas por colchetes [], parênteses () ou chaves, como no exemplo abaixo.

```
>>> meses = ['jan', 'fev', 'mar',
'abr', 'mai', 'jun', 'jul',
'ago', 'set', 'out', 'nov', 'dez']
>>> meses
['jan', 'fev', 'mar', 'abr', 'mai', 'jun', 'jul', 'ago', 'set',
'out', 'nov', 'dez']
```

3.4 Múltiplas declarações em uma linha

Podem-se fazer múltiplas declarações em uma única linha desde que separadas por ponto e vírgula. Observe o exemplo:

```
a = 10; b = 20; c = 'multiplos comandos'
>>> a
10
>>> b
20
>>> c
'multiplos comandos'
```

3.5 Separadores e delimitadores

Um separador é um símbolo que separa outros símbolos. O branco é o separador mais utilizado, outros separadores são a vírgula utilizada em listas, e o ponto usado para separar a parte inteira da fracionária em um número.

As aspas simples e duplas são delimitadores utilizados em literais e cadeias de caracteres.

Os blocos de comandos são delimitados em Python pela indentação (**recuo**, derivado da palavra em inglês indentation, também grafado nas formas **identação** e **endentação**)

Exemplo:

```
>>> for i in range(0,2):
        print i

0
1
```

3.6 Comentários

Os comentários são usados para dar mais clareza aos programas. Todo comentário é ignorado pelo compilador da linguagem. Seu uso é muito recomendado para facilitar o entendimento do programa, especialmente quando outros programadores tiverem que entender o código.

Os comentários em Python são precedidos do caractere hash #. Por exemplo:

```
>>> #isto é um comentário
>>> 2 + 2
4
>>> 2 + 3 #isto é um comentário na linha de comando
5
```

3.7 Documentação (Docstring)

Docstrings são tipos especiais de comentários utilizados para documentação de programas, funções ou classes. Indica-se um doctstring dentro de um programa por aspas triplas sendo que principal diferença entre os docstrings e os comentários não se dá somente na parte estética, com esse recurso o usuário pode acessar todos os textos docstring através do modulo help. Não se deve usar caracteres especiais no docstring.

```
'''isso
eh uma doc string
voce pode escrever como quiser
ate terminar a declaracao'''

#isso é um comentário
```

Capítulo 4

Tipos Primitivos de Dados

Os tipos de dados estão diretamente relacionados com a natureza da informação e da maneira pela qual o computador manipula as informações. Neste capitulo trataremos dos tipos nativos caracterizados por já estarem inclusos no núcleo da linguagem Python. O Python utiliza a **Tipagem dinâmica** o que significa que o próprio programa define os tipos das variáveis conforme seu uso sem a necessidade do programador declarar previamente.

Os tipos primitivos de dados podem ser simples ou estruturados.

4.1 Tipos Simples de Dados

Todos os tipos simples de dados em Python são **imutáveis** e no total são quatro: numéricos, alfanumerico, lógicos e vazio.

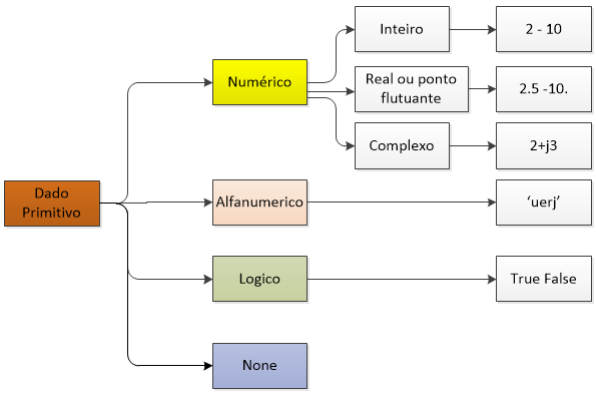


Figura 4.1: Tipos de Dados Simples

4.1.1 Numéricos

Os dados numéricos são, como o nome já mostra, os números de forma geral.
Podem ser:

•**Inteiro** Toda e qualquer informação numérica que pertença ao conjunto dos números inteiros. Os números inteiros são representados em precisão simples ou em precisão dupla. Os Inteiros simples utilizam 32 bits para representar os valores e englobam a faixa de -2147483648

até 2147483647, mas podem ser maiores, dependendo do processador e do sistema operacional. Os Inteiros longos são outra classe de inteiros de abrangência ilimitada, ou limitada apenas pela memória da máquina.

A conversão de um valor para seu representante inteiro é feita através do comando: int(x), onde x é uma variável ou um valor.

```
>>> a=3.7
>>> int(a)
3
```

•**Reais ou ponto flutuante** Toda e qualquer informação numérica que pertença ao conjunto dos números reais. São caracterizados pela existência do ponto decimal. Os números reais são representados em precisão dupla com 64 bits. Podem ser escritos das seguintes formas:

1.5, -47.2 ou 1E3 (onde a letra E significa 10 elevado a).

O exemplo 1E3 representa o valor 1000.0 e significa 1 vezes 10 elevado a 3.

A conversão de um valor para seu representante real, também chamado de ponto flutuante ou float, é feita através do comando: float(x), onde x é uma variável ou um valor.

Por exemplo:

```
>>> c=3
>>>float(c)
3.0
```

•**Complexos ou números imaginários**

São descritos por dois valores reais, a parte real e a parte imaginária na forma (real +IMAG J). Neste caso, o número i () é designado pela letra j.

Por exemplo: (2+3j)

Para se extrair uma parte de um numero complexo use: a.real ou a.imag. Por exemplo:

```
>>> a=2+3j
>>> a.real
2.0
>>> a.imag
3.0
```

4.1.2 Lógicos (ou Booleanos)

Os valores dessa classe se resumem a dois resultados:

•**False**

É tudo aquilo que corresponde ao valor 0 na memória, ou seja listas, strings, dicionários vazios ou até mesmo o 0.

•**True**

Tudo aquilo que não for **False**

•**Conversão para dados lógicos**

O comando bool(x) converte um dado para lógico

Por exemplo:

```
>>> a=1
>>> b=0
>>> bool(a)
True
>>> bool(b)
False
```

4.1.3 Alfanuméricos (ou string)

Valores alfanuméricos também chamados de literais são objetos de tipo imutável do Python e que possuem noção de ordenação. São constituídos de caracteres e declarados entre apóstrofos ou aspas.

Os dados literais denotam um conjunto de caracteres da tabela ASCII (ou UNICODE), ou seja:

- As letras maiúsculas e minúsculas;
- Os dígitos de 0 a 9;
- Os caracteres especiais, como os de pontuação, de operação etc.;
- Os caracteres de controle e que não são imprimíveis.

Por exemplo:

```
>>> a = "este e um exemplo"
>>> a
'este e um exemplo'
```

A conversão de um valor para literal, também chamada de texto ou string, é feita através do comando construtor da classe (str(x), onde x é um valor ou uma variável relacionada a um valor imprimível).
Por exemplo:

```
>>> a = 3.5
>>> b = str(a)
>>> b
'3.5'
```

4.1.4 Vazio

Esse tipo de dado é representado pelo valor None e não representa nada e não aceita operações comuns. Atenção: O **N** de None é maiúsculo!!!

```
>>> a=None
>>> a*3
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
a*3
TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
```

4.2 Tipos Estruturados de Dados

Os tipos estruturados de dados também chamados de compostos são aqueles constituídos de um agrupamento, **ordenado ou não**, dos tipos primitivos de dados. São eles: listas, conjuntos,tuplas e dicionários.

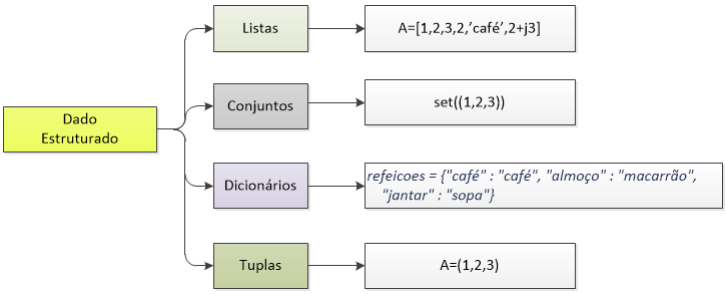


Figura 4.2: Tipos Estruturados de Dados

4.2.1 Listas

Uma lista (ou list) é um conjunto ordenado de valores, onde cada valor é identificado por um índice. Os valores que compõem uma lista são chamados elementos.Nas listas os elementos podem ser valores de quaisquer natureza,repetidos ou não e, não necessariamente homogêneos. Os elementos são separados por virgula com chaves [ ] no inicio em fim da lista.Uma lista é um objeto mutável.

```
notas=[1,2,3,'uerj']
```

4.2.2 Conjuntos

Um conjunto (ou set) é um conjunto ordenado de valores **sem valores repetidos** de forma similar as listas. Os valores que compõem um conjunto podem ser de qualquer natureza, porém deve-se ter atenção que ao converter uma lista em um conjunto pode-se alterar a ordem dos elementos.

```
>>> a=[1,2,3,2,'uerj']
>>> a
[1, 2, 3, 2, 'uerj']
>>> b=set(a)
>>> b
set([1, 2, 3, 'uerj'])
```

4.2.3 Tuplas

É um conjunto ordenado de valores de forma similar as listas diferencia desta por ser um objeto **imutável**. Em uma tupla, os elementos são separados por vírgula e não possuem chaves [ ] no inicio e no fim.

```
notas=1,2,3,"uerj"
```

Pode-se usar tuplas para fazer o *swap* (intercambio de valores) entre duas variáveis sem a necessidade de se usar uma variável temporária.
Por exemplo:

```
>>> a,b=2,3
>>> a
2
>>> b
3
>>> b,a=a,b
>>> a
3
>>> b
2
```

4.2.4 Dicionários

Dicionários são conjunto ordenado de pares de valores. Os elementos de um dicionário aparecem em uma lista separada por vírgulas e com colchetes { } no inicio e fim do dicionário. Cada entrada contêm um índice e um valor separado por dois-pontos. Em um dicionário, os índices são chamados de chaves, então os elementos são chamados de pares chave-valor.

```
>>> dic = {"joao":100,"maria":150}
```

Capítulo 5

Expressões Aritiméticas, Lógicas e Relacionais

5.1 Aritméticas

São usados para o desenvolvimento de operações aritméticas e cálculos matemáticos, em geral. São escritos linearmente usando a notação matemática.

Os operadores são:

+ Soma

```
>>> 2 + 3
5
```

- Subtração

```
>>> 3 - 5
-2
```

*** Multiplicação**

```
>>> 2 * 3
6
```

/ Divisão Real

```
>>> 2 / 3
0.666666666666
```

% Resto da divisão inteira

```
>>> 3 % 2
1
>>> 2 % 3
2
```

**** Exponenciação**

```
>>> 2 ** 3
8
```

// Divisão inteira

```
>>> 3 // 2
1
>>> 2 // 3
0
```

O Python calcula 0**0 sendo igual a 1.
Ao efetuar operações devem-se observar as seguintes regras:

- Não pode existir nenhuma operação implícita;
- Para agrupar as operações matemáticas utiliza-se exclusivamente o parêntese;
- Dois operadores podem aparecer juntos, porém dificulta sua legibilidade. Deve-se utilizar o parêntese para separar;
- Pode-se atribuir um valor a mais de uma variável.

Por exemplo:

```
>>> a = b = c = 3 #a = 3, b = 3, c = 3
>>> x, y = 0, 1 #x = 0, y = 1

>>> y = 1
>>> y += 1 #y = y + 1
>>> y
2
>>> y *= 3 #y = y * 3
>>> y
6
>>> y /= 3 #y = y / 3
>>> y
2
>>> y -= 1 #y = y - 1
>>> y
1
>>> y // 2 #y = y // 2
>>> y
0
>>> y ** 0 #y = y ** 0
>>> y
1
```

5.1.1 Hierarquia dos Operadores

1º	Parêntesis
2º	Exponenciação
3º	* / % //
4º	+ -

Em caso de **mesma hierarquia** resolve-se da **esquerda para a direita**, conforme na matemática.

Observe o exemplo:

```
>>> #Veja a importancia do uso dos parênteses
>>> (2 + 3)/5
1
>>> # (2 + 3)/5 = 5/5 = 1
>>> 2 + 3/5
2
>>> #2 + 3/5 = 2 + 0 = 2 Note que o Python interpretará uma divisão
como float quando pelo menos um dos valores for float
>>> 2 + 3.0/5
2.6
```

5.2 Operações a Nível de Bit

O Python permite ao programador deslocar bits alocados na memória. Essa operação se dá através do operador >> ou <<, para deslocar os bits a direita e a esquerda, respectivamente.

```
>>> a = 8
>>> a<<2
32
>>> #Em binário a = 1000 e 32 = 100000, como podem notar ele deslocou
os bits de a, 2 casas para a esquerda
>>>
```

5.3 Erros de Float Point

O Python não consegue fazer todos os cálculos de forma 100% precisa, isso se deve ao fato da programação induzir determinados erros. Note o exemplo abaixo:

```
>>> .1 + .2
0.30000000000000004
```

Esse erro ocorreu porque a maioria das frações decimais não pode ser representada de forma correta em binário, sendo então representada de forma aproximada.

5.4 Lógicas

São expressões cujos operadores são lógicos e cujos operandos são relações tendo como resposta um valor booleano (**True** ou **False**).

Os operadores relacionais são:

== Verifica se dois operadores são iguais.

```
>>> 2 == 3
False
```

!= Verifica se o primeiro operador é diferente do segundo.

```
>>> 2 != 3
True
```

> Verifica se o operando da esquerda é maior que o da direita.

```
>>> 3 > 2
True
```

>= Verifica se o operando da esquerda é maior ou igual que o da direita.

```
>>> 2 >= 3
False
```

< Verifica se o operando da esquerda é menor que o da direita.

```
>>> 2 < 3
True
```

<= Verifica se o operando da esquerda é menor ou igual do que o da direita.

```
>>> 2 <= 3
True
```

Os operadores lógicos são:

***and* Retorna True apenas quando os dois operandos possuam valor True.**

```
>>> 2 > 3 and 3 < 4
False
```

***or* Retorna True caso um dos operandos possuam valor True.**

```
>>> 2 > 3 or 3 < 4
True
```

***in* Verifica a participação como membro de um elemento.**

```
>>> 2 in [2,3,4]
True
```

***is* Verifica se os dois operandos são iguais.**

```
>>> 2 is 3
False
```

***not* Inverte o valor lógico de uma condição.**

```
>>> 1 not in [2,3,4]
True
```

5.5 Exercícios Resolvidos

1. De acordo com os dados:

```
>>> a=5
>>> b=3
>>> c=2
>>> d=1
>>> e=0
```

Escreva o tipo de saída de cada uma das execuções abaixo:

a)
```
>>> a - b + c + d + e == a - (b + c + d + e)
False
```

b)
```
>>> not(a) is True
True
```

c)
```
>>> a > b
True
```

d)
```
>>> b < c
False
```

e)
```
>>> bool(a)
True
```

f)
```
>>> bool(e)
False
```

g)
```
>>> bool(a) or bool(e)
True
```

h)
```
>>> bool(a) and bool(e)
False
```

i)
```
>>> bool(a) or bool(e) and bool(a)
True
```

j)
```
>>> not bool(b)
False
```

k)
```
>>> not bool(b) != True
True
```

l)
```
>>> True != not(bool(b))
SyntaxError: invalid syntax
```

m)
```
>>> True != not bool(b)
SyntaxError: invalid syntax
```

n)
```
>>> is a == True
SyntaxError: invalid syntax
```

o)
```
>>> True != not(bool(b)))
True
```

Capítulo 6

Entrada e Saída

Os comandos de Entrada e Saída são usados respectivamente para receber dados dos usuário e mostrar dados para o usuário.

6.1 Comandos de Entrada

A versão 2.7.3 do Python aceita os comandos:

- raw_input - usada para entrada de dados do tipo texto (ou string)

  Variável = raw_input(‘mensagem’)

  Neste comando tudo é recebido como texto. Desse modo, caso o usuário queira receber um número é necessário transformar de texto para número (int ou float).

  Este tipo de declaração é mais estável por evitar erros e possibilitar que o programador verifique se o dado de entrada é válido antes de utilizá-lo. Contudo para a realização dos testes será exigido um esforço maior por parte do programador.

```
>>> a = raw_input('Digite o nome ')
Digite o nome Python
>>> a
'Python'
```

- input - usada para entrada de dados de uma forma geral.

  Variável = input(‘mensagem’)

  Esta declaração é utilizada principalmente para a entrada de dados numéricos, devido sua necessidade de conhecimento prévio do usuário do tipo de dado de entrada e da forma de declaração feita em Python.

```
>>> a = input('Digite um numero ') #dado numerico
Digite um numero 2
>>> a = input('Digite um numero ') #dado numerico real
Digite um numero 2.5
>>> a
```

```
2.5
>>> nome = input('Digite o nome ')
Digite o nome 'python' #observe as aspas
>>> nome
'python'
```

Os comandos de entrada podem também ser utilizados para aguardar uma ação do usuário como podemos perceber no exemplo a seguir:

```
>>> raw_input("precione qualquer tecla para finalizar o programa")
precione qualquer tecla para finalizar o programa
```

6.2 Comandos de Saída

A impressão de alguma mensagem na tela é indicada da seguinte forma:
print "mensagem", variável1, ..., variaveln, "mensagem2", variável, ..., "mensagem"
Exemplo:

```
>>> nota1 = 7.5
>>> nota2 = 8.5
>>> nome = 'Antonio'
>>> print 'O aluno', nome, 'teve notas', nota1, nota2
O aluno Antonio teve notas 7.5 8.5
```

Podem-se agrupar várias mensagens tipo texto usando o operador +. Por exemplo:

```
>>> nota1 = 7.5
>>> nota2 = 8.5
>>> nome = 'Antonio'
>>> print 'O aluno', nome, 'P1 ' + str(nota1), 'P2 ' + str(nota2)
O aluno Antonio P1 7.5 P2 8.5
```

Forma errada de declarar:

```
>>> print ('as letras a, b, c correspondem aos valores', a, b, c, 'respectivamente')
('as letras a, b, c, correspondem aos valores', 1, 2, 3, 'respectivamente')
```

Pode-se perceber que no caso acima o interpretador exibe de forma errada. Esse erro ocorre porque o Python interpreta os dados como se fosse uma tupla com elemento 0 do tipo string, elementos 1, 2, 3 do tipo numérico e o elemento 4 do tipo string.

6.3 Formatação para Exibição de Dados

O programador pode formatar a exibição dos valores de variáveis indicando uma "máscara" de acordo com seu tipo.

Declara-se que um valor será exibido naquela posição através do parâmetro %x, onde x é o código característico para sua formatação.

Para comandos de saída utiliza-se rotineiramente o tipo string devido à facilidade da conversão automática.

Segue aqui uma lista de valores que x pode assumir com seus respectivos tipos:

%d → Utilizado para a declaração de um número inteiro.

```
>>> print 'joão tem %d anos e %d bolinhas de gude' %(15,10)
joão tem 15 anos e 10 bolinhas de gude
```

%e → Utilizado para a declaração de um número float que será convertido para a forma exponencial.

```
>>> print 'joão tem %e dias de vida' %(9)
joão tem 9.000000e+00 dias de vida
>>> print 'joão tem e dias de vida' %(121)
joão tem 1.210000e+02 dias de vida
```

%f → Utilizado para a declaração de um número float.

```
>>> print 'um ano tem %f dias por mes' %(365/12)
um ano tem 30.000000 dias por mes
>>> print 'um ano tem %f dias por mes' %(365.0/12)
um ano tem 30.416667 dias por mes
```

%c → Utilizado para a declaração de um único caractere.

```
>>> print 'a letra sorteada foi: %c' %('a')
a letra sorteada foi: a
```

%s → Utilizado para a declaração de uma string.

```
>>> print '%s, %s' %('string de exemplo', 10)
string de exemplo, 10
```

%g → Utilizado para a declaração de um número float que será convertido para a forma exponencial somente se sua precisão for maior do que 4.

```
>>> print 'joão tem %g dias de vida' %(121)
joão tem 121 dias de vida
>>> print 'joão tem %g dias de vida' %(9)
joão tem 9 dias de vida
>>> print 'joão tem %g dias de vida' %(10349)
joão tem 10349 dias de vida
>>> print 'joão tem %g dias de vida' %(10000009)
joão tem 1e+07 dias de vida
```

6.4 Números Octais ou Hexadecimais

Dentro de um programa os números podem estar em base octal ou hexadecimal por questões de conveniência. Dessa forma, para imprimir um numero octal na base decimal basta incluir um 0 na frente desse numero, ou incluir um 0X em casos de hexadecimal.

Vale ressaltar que o número precisa ter no mínimo 3 algarismos, para a conversão em octal. Não há restrições para conversões em hexadecimal.

Exemplo:

```
>>> print 0100
64
>>> print 0X100
256
```

6.5 Exercícios Resolvidos

1. Escreva um programa em Python que leia dois valores e imprima o valor de a/b. Exemplo 7.0/2 = 3.5

   ```
   a = input('A:')
   b = input('B:')
   print "a/b=", a/b
   ```

2. Escreva um programa em Python que leia três valores e imprima a média aritmética desses valores.

   ```
   n1 = input('n1:')
   n2 = input('n2:')
   n3 = input('n3:')
   media = (n1+n2+n3)/3.0

   print 'media: ', media
   ```

3. Escreva um programa em Python que leia os valores em horas, minutos e segundos e exiba o resultado convertido para segundos.

   ```
   h = input('Hora: ')
   min = input('Minuto: ')
   seg = input('Segundo: ')
   total = h*3600 + min*60 + seg
   print "total: ", total
   ```

4. Escreva um programa em Python que leia um valor em segundos e exiba a conversão para horas, minutos e segundos.

```
tempoSeg = input('tempo em segundos:')
hora = tempoSeg // 3600
minuto = tempoSeg % 3600 // 60
segundo = tempoSeg % 3600 % 60

print  hora, ':', minuto, ':', segundo
```

5. Escreva um programa em Python que leia um valor em centímetros e o imprima em polegadas. (1 pol = 2,54 cm)

```
cm = input('entre com um valor em cm ')
pol = 2.54 * cm

print "Voce tem %s polegadas" % (pol)
```

6. Escreva um programa em Python que leia o raio de um circulo e imprima o comprimento e área do circulo.

```
raio = input('Entre com um raio: ')
comprimento = 2 * 3.14 * raio
area = 3.14 * raio ** 2

print "O comprimento do circulo eh:", comprimento
print "A area do circulo eh:", area
```

7. Escreva um programa em Python que leia dois valores e imprima o resto da divisão do primeiro pelo segundo e o quociente da divisão.

```
a = input('Entre com o dividendo: ')
b = input('Entre com um divisor: ')

print "Seu resto eh: ", a % b
print "Seu quociente eh: ", a//b
```

8. Escreva um programa em Python que calcule o comprimento de uma onda, sendo inserida sua frequência e sua velocidade. Lembrando que v = λ.f.

```
v = input('Insira a velocidade: ')
f = input('Insira a frequencia: ')
comprimento = v/f #lambda eh uma palavra reservada

print "O comprimento da onda eh", comprimento, "UC"
```

9. Escreva um programa em Python que calcule a hipotenusa de um triangulo retângulo.

```
b = input('Entre com o cateto: ')
c = input('Entre com o cateto: ')
a = float(b ** 2 + c ** 2)
a **= 1/2.0 #Lembrando que DEVE ser 2.0 no denominador

print a
```

10. Escreva um programa em Python que simule um caixa eletrônico. Para isso será lido o valor em real e o programa exibira a quantidade de cada nota que será entregue ao cliente. O nosso caixa eletrônico contém as notas de 50, 20, 10 5, 2 e 1.

```
saque = input('Entre com o valor do saque: ')
nota50 = saque // 50
resto  = saque % 50
nota20 = resto // 20
resto  = resto % 20
nota10 = resto // 10
resto  = resto % 10
nota5  = resto // 5
resto  = resto % 5
nota2  = resto // 2
resto  = resto % 2

print  'Nota 50:', nota50, 'Nota 20', nota20, 'Nota10', nota10,\
'Nota 5:', nota5, 'Nota 2:', nota2, 'Nota 1:', resto
```

6.6 Exercícios Propostos

1. Escreva um programa em Python que leia dois valores **a** e **b** nessa ordem, e caso **a**>**b** seja verdade, o valor de **b** se torna o valor de **a** e o valor de **a** se torna o de **b** e imprime **a**, **b** e "valores trocados", caso contrário imprime "valores não alterados".

2. Escreva um programa em Python que calcule a velocidade média de um corpo, sendo dados Δs e Δt. Lembrando $vm = \frac{\Delta s}{\Delta t}$.

6.7 Para Saber Mais

Para maiores informações sobre os comandos de entrada:

```
>>> help("input")
>>> help("raw_input")
```

Capítulo 7

Estruturas de Decisão

Esses comandos analisam e executam uma única vez um bloco de instruções de acordo com o valor verdadeiro de uma condição. São usados quando queremos executar desvios no programa.

7.1 Estruturas de Decisão Simples

if condição:
Essa estrutura executa um bloco de instruções caso a condição estabelecida retorne True.
if condição:
    Comando ou bloco de comandos
Continuação do código

7.2 Estruturas de Decisão Compostas

elif → É verificado caso a condição do if retorne False. Caso retorne True, executa o bloco de instruções e ignora todos os elif e else abaixo.
else → É executado caso todos o if e todas as condições de elif acima dele, e a ele relacionado, retornem falso.
Exemplo:

```
if condição:
    Comando ou bloco de comandos
elif condição:
    Comando ou bloco de comandos
elif condição:
    Comando ou bloco de comandos
else:
    Comando ou bloco de comandos
Continuação do código
```

7.3 Exercícios Resolvidos

1. Escreva um programa em Python que leia dois números inteiros A e B da entrada padrão (teclado) e retorne o quociente da divisão entre A e B. O programa deve verificar previamente à divisão, se o valor de B é diferente de zero.

```
a = float(raw_input("Entre com dividendo: "))
b = float(raw_input("Entre com divisor: "))

if b == 0:
        print("Impossivel dividir por 0")
else:
        c = a/b
        print "O quociente da divisao de {0} por {1} eh {2}"\
        .format(a, b, c);
```

2. Escreva um programa em Python que receba o sexo e a idade de uma pessoa. Se a pessoa for do sexo feminino e tiver menos de 25 anos, imprimir o sexo, a idade e a mensagem: ACEITA. Caso contrário, imprimir o sexo, a idade e a mensagem não aceita. (Considerar para o Sexo as letras F, f, M ou m).

```
sexo = raw_input("Entre com seu sexo(F/M): ").upper()
idade = int(raw_input("Entre com sua idade: "))

if(sexo == 'F' and idade < 25):
        print "Sexo {0}, {1} anos, ACEITA".format(sexo, idade)
elif(sexo == 'M'):
        print "Sexo {0}, {1} anos, NAO ACEITA".format(sexo, idade)
```

3. Escreva um programa em Python que lê 3 valores, verifica se é um triangulo, caso positivo informa se é equilátero, isósceles ou escaleno e seu perímetro.

```
a = float(raw_input("Entre com o primeiro valor: "))
b = float(raw_input("Entre com o segundo valor: "))
c = float(raw_input("Entre com o terceiro valor: "))

if((abs(b - c) < a < b + c) and (abs(a - c) < b < a + c) and\
(abs(a - b) < c < a + b)):
        if(a == b == c):
                print "Triangulo equilatero"
        elif(a == b or a == c or b == c):
                print "Triangulo isosceles"
        else:
                print "Triangulo ecaleno"

        perimetro = a + b + c
        print "O seu perimetro eh {0}".format(perimetro)
```

```
else:
        print "Nao eh um triangulo"
```

4. Um endocrinologista deseja controlar a saúde de seus pacientes e, para isso se utiliza de um índice de massa corporal (IMC). Sabendo-se que o IMC é calculado através da fórmula abaixo:

   $IMC = \frac{Peso}{Altura^2}$

   Onde o Peso é dado em Kg e a Altura é dada em metros. Escreva um programa em Python que apresente o nome do paciente e sua faixa de risco, baseando-se na seguinte tabela:

IMC	Faixa de Risco
Abaixo de 20	Abaixo do Peso
A partir de 20 até 25	Normal
Acima de 25 até 30	Excesso de Peso
Acima de 30 até 35	Obesidade
Acima de 35	Obesidade Mórbida

```
nome = raw_input("Entre com seu nome: ")
peso = float(raw_input("Entre com seu peso: "))
altura = float(raw_input("Entre com sua altura: "))
imc = float(peso/(altura ** 2))
faixaRisco = ""

if(imc < 20):
        faixaRisco = "Abaixo do Peso"
elif(imc <= 25):
        faixaRisco = "Normal"
elif(imc <= 30):
        faixaRisco = "Excesso de Peso"
elif(imc <= 35):
        faixaRisco = "Obesidade"
else:
        faixaRisco = "Obesidade Morbida"

print "Nome: {0}\nFaixa de Risco: {1}".format(nome, faixaRisco)
```

5. Escreva um programa em Python para o cálculo das raízes de uma equação do segundo grau. Seu programa não deve calcular raízes imaginárias.

```
a = float(raw_input("Entre com o coeficiente do termo de grau 2: "))
b = float(raw_input("Entre com o coeficiente do termo de grau 1: "))
c = float(raw_input("Entre com o termo independente: "))

delta = (b**2 - 4*a*c)

if delta >= 0:
```

```
        x1 = (-b + (delta ** (1/2.0)))/(2.0*a)
        x2 = (-b - (delta ** (1/2.0)))/(2.0*a)
        print("Suas raizes sao: {0} e {1}").format(x1, x2)
else:
        print("Raizes imaginarias")
```

6. Escreva um programa em Python que informe o dia seguinte. Note você deverá ler e informar dia, mês e ano e considerar se ele é ou não bissexto. Suponha que as datas serão válidas.

```
a = int(raw_input("Digite o dia: "))
b = int(raw_input("Digite o mes: "))
c = int(raw_input("Digite o ano: "))
bissexto = False

if(a == 30 and (b == 4 or b == 6 or b == 9 or b == 11)):
        a = 1
        b += 1
elif(a == 31):
        if(b == 12):
                a = 1
                b = 1
                c += 1
        else:
                a = 1
                b += 1
elif(b == 2):
        if(c % 4 == 0):
                bissexto = True
                if(c % 100 == 0):
                        if(c % 400 == 0):
                                bissexto = True
                        else:
                                bissexto = False
        if(bissexto):
                if(a == 28):
                        a += 1
                elif(a == 29):
                        a = 1
                        b += 1
        else:
                if(a == 28):
                        a = 1
                        b += 1
else:
        a += 1
```

```
print "Proximo dia: {0}/{1}/{2}".format(a, b, c)
```

7. O Ceil retorna o menor inteiro maior que um número. Escreva um programa em Python que o simule e trate o caso de inserir um número inteiro.

```
num = float(raw_input("Entre com um numero: "))

if(num > int(num)):
        num = int(num) + 1
        print num
else:
        print num
```

8. Um guerreiro nunca luta com um exército que tenha um número menor de tropas que o seu. Escreva um programa em Python que leia dois valores e mostre o número de tropas do guerreiro, o do oponente e sua diferença.

```
tropa1 = int(raw_input("Entre com o numero da primeira tropa: "))
tropa2 = int(raw_input("Entre com o numero da segunda tropa: "))

if(tropa1 >= tropa2):
        diff = tropa1 - tropa2
        print "Tropas do guerreiro: {0}\nTropas do oponente: {1}\
        \nDiferenca: {2}".format(tropa2,tropa1,diff)
else:
        diff = tropa2 - tropa1
        print "Tropas do guerreiro: {0}\nTropas do oponente: {1}\
        \nDiferenca: {2}".format(tropa1,tropa2,diff)
```

9. Escreva um programa em Python para ler duas notas de um aluno e imprimir a sua situação de acordo com os critérios da UERJ.

```
nota1 = float(raw_input("Entre com sua primeira nota: "))
nota2 = float(raw_input("Entre com sua segunda nota: "))
media = (nota1 + nota2)/2.0

if(media >= 7):
        print "Aprovado"
elif(media >= 4):
        print "Prova Final"
else:
        print "Reprovado"
```

10. Escreva um programa em Python para ler a idade de uma pessoa e imprimir sua situação de acordo com os critérios abaixo:

- Idade <= 0 → ERRO
- 1 <= idade <= 3 → BEBE
- 4 <= idade <= 11 → CRIANÇA
- 12 <= idade <= 17 → ADOLESCENTE
- 18 <= idade <= 30 → JOVEM
- 31 <= idade <= 64 → ADULTO
- idade >= 65 → IDOSO

```
idade = int(raw_input("Entre com uma idade: "))

if(idade <= 0):
        print "Erro"
elif(idade <= 3):
        print "Bebe"
elif(idade <= 11):
        print "Crianca"
elif(idade <= 17):
        print "Adolescente"
elif(idade <= 30):
        print "Jovem"
elif(idade <= 64):
        print "Adulto"
else:
        print "Idoso"
```

7.4 Exercícios Propostos

1. Escreva um programa em Python que leia um número e imprima a palavra "PAR"se ele for par e a palavra "ÍMPAR"se ele for ímpar. Além disso, imprima a palavra "MULT3"se ele for múltiplo de 3.

7.5 Para Saber Mais

```
>>> help("if")
```

Capítulo 8

Estruturas de Repetição

São usadas para repetir um conjunto de instruções.

8.1 Comando while

Repete um conjunto de instruções enquanto uma condição for verdadeira. Devemos estar atentos para que a condição se torne falsa durante a execução para interromper a repetição. A condição é testada antes de entrar na estrutura de repetição.

FLAG ou sentinela - é um dado que é utilizado como verificador de continuidade de um laço de repetição. É recomendado ao programador, na maioria dos casos, a utilização de um flag no lugar de uma comparação a fim de aperfeiçoar o algoritmo.

8.1.1 Estrutura while Simples

```
while condição:
    Comando ou bloco de comandos;
Continuação do código
```

8.1.2 Estrutura while Composta

A estrutura while pode vir acompanhada de um else, que será iniciado no momento em que a condição do while retorne False. Contudo devemos ficar atentos, pois o else, ao contrário do while, será executado somente uma vez.

```
while condição:
    Instrução
    ...
    Instrução
else:
    Instrução
    ...
    Instrução
Continuação do código
```

8.1.3 Laços Infinitos

Ocorre quando o comando while é executado eternamente. Para encerrar a execução de um laço infinito a força, devemos pressionar ctrl+c duas vezes na Shell do Python.

8.2 Comando for

Percorre os itens de objetos e sequências de objetos onde exista a ideia de ordem atribuindo os valores à variável e executando o bloco de comandos após cada atribuição.

for variável in objeto sequencial:
    Comando1
    ...
    Comando n
Continuação do código

8.2.1 O Gerador de Listas range

Essa função gera uma lista cujos elementos são uma sequência que possui início e fim e que seguem uma determinada razão ou passo. Uma observação importante é que o intervalo percorrido pela função range é que o intervalo é fechado no início e aberto no fim.

```
range (inicio , fim , passo)
```

8.3 O Comando break

Esse comando finaliza o laço de repetição mais interno que o envolve, passando para o próximo comando fora do laço. Com base nessa ideia podemos concluir que o programa:

```
while True:
        b = 1
        a = 1
        while a == 1:
                print('entrou no laço')
                if a == 1:
                        break
                print('não vou aparecer')
        print('to aparecendo pro usuário')
        b = b + 1
        if b == 2:
                break
        print('não quero ficar em loop infinito')
        print('não vou aparecer também')
print('programa finalizado')
```

Produzirá o seguinte resultado:

```
>>>
entrou no laço
to aparecendo pro usuário
programa finalizado
```

8.4 O Comando continue

O comando continue passa para o próximo passo do laço mais interno.
Dessa forma o programa:

```
a = b = - 1
while True:
        while b != 22:
                print(a)
                print(b)
                if a == -1:
                        a = a +1
                        continue
                b = 22
        else:
                print(b)
                break
print('programa finalizado')
```

Produzirá o seguinte resultado:

```
>>>
-1
-1
0
-1
22
programa finalizado
```

8.5 A Construção pass

A Construção pass não faz nada. Ela pode ser usada quando a sintaxe exige um comando, mas a semântica não requer nenhuma ação.

```
for i in range(0, 10):
        pass
print i
```

Produzirá o seguinte resultado:

```
>>>
9
>>>
```

8.6 Exercícios Resolvidos

1. Escreva um programa em Python que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.

```
nota = float(raw_input("Digite uma nota entre zero e dez: "))

while(nota < 0 or nota > 10):
        print "Nota fora do intervalo"
        nota = float(raw_input("Digite uma nota entre zero e dez: "))
```

2. Escreva um programa em Python que mostre os números entre 1000 e 2000 que divididos por 11 dão resto 5.

```
for i in range(1001, 2000):
        if((i % 11) == 5):
                print i
```

3. Escreva um programa em Python que leia a nota de um número indeterminado de alunos, perguntando ao final se ele deseja ou não continuar a executar o programa.

```
cont = 1

while True:
        nota = float(raw_input("Entre com a nota do aluno {0}: ".format(cont)))
        cont += 1

        resp = raw_input("Entre com 0 se deseja parar o programa: ")

        if(resp == '0'):
                break
```

4. Escreva um programa em Python que leia um valor n inteiro e positivo e x real e que calcule as seguintes somas:

   - $S1 = x + \frac{x}{2} + \frac{x}{3} + \frac{x}{4} + ... + \frac{x}{n}$
   - $S2 = x + \frac{x}{2!} + \frac{x}{3!} + \frac{x}{4!} + ... + \frac{x}{n!}$
   - $S3 = x - \frac{x}{2!} + \frac{x}{3!} - \frac{x}{4!} + ... \pm \frac{x}{n!}$
   - $S4 = 1 + \frac{x}{2!} + \frac{x}{4!} + \frac{x}{6!} + ... + \frac{x}{n!}$

```
n = int(raw_input("Entre com um valor inteiro e positivo n: "))
x = float(raw_input("Entre com um valor real x: "))
s1 = s2 = s3 = 0.0
s4 = 1.0
fat = 1
```

```
for i in range(1, n+1):
        fat *= i
        s1 += x/i
        s2 += x/float(fat)

        if((i % 2) == 0):
                s3 -= x/float(fat)
                s4 += x/float(fat)
        else:
                s3 += x/float(fat)

print "S1 = {0}\nS2 = {1}\nS3 = {2}\nS4 = {3}".format(s1,s2,s3,s4)
```

5. Escreva um programa em Python que exiba se um número X, fornecido pelo usuário, é ou não primo.

```
num = int(raw_input("Entre com um numero: "))
primo = True

for i in range(2, num):
        if((num % i) == 0):
                primo = False
                break

if(primo):
        print "Eh um numero primo"
else:
        print "Nao eh um numero primo"
```

6. Escreva um programa em Python que leia o valor e o multiplique por 3 até que ele seja maior que 100. Seu programa deverá tratar os casos em que ocorre loop infinito.

```
num = float(raw_input("Entre com um numero: "))
cont = 0

if(num <= 0):
        print "Loop infinito"
else:
        while(num < 100):
                num *= 3
                cont += 1
        print "Vezes que o numero foi multiplicado: {0}".format(cont)
```

7. Em um frigorífico existem 200 bois. Cada boi tem uma identificação contendo um número maior que 0 e seu peso. Escreva um programa em Python que leia as identificações e mostre o número do boi mais pesado.

```
idPesado = -1
maiorPeso = -1

for i in range(0, 200):
        boi = int(raw_input("Entre com a identificacao do boi: "))
        peso = float(raw_input("Entre com o peso do boi: "))

        if(peso > maiorPeso):
                maiorPeso = peso
                idPesado = boi

print "A identificacao do boi mais pesado eh: {0}".format(idPesado)
```

8. Escreva um programa em Python que calcule e exiba a média aritmética dos números lidos entre 13 e 73. O flag é um número negativo.

```
num = float(raw_input("Entre com um numero: "))
cont = 0
soma = 0.0

while num >= 0:
        if(num > 13 and num <73):
                cont += 1
                soma += num
num = float(raw_input("Entre com um numero: "))

if(cont > 0):
        media = soma/cont
        print "Media: {0}".format(media)
else:
        print "Media: 0"
```

9. Hugo, um aluno muito distraído, inseriu o seguinte código e obteve um resultado inesperado:

```
>>> import math
>>> math.cos(4)
-0.6536436208636119
```

Confuso Hugo indagou o porque de o cosseno ter dado negativo, mesmo 4° estando no primeiro quadrante. Como você explicaria para Hugo o que aconteceu?

Resposta:
O problema se deu pelo fato de Hugo ter acreditado que o argumento de entrada era dado em graus. Na realidade Hugo inseriu um valor próximo de 229°, o que justifica o cosseno negativo.

10. Escreva um programa em Python para ler o nome, sexo e duas notas dos alunos de uma turma até que seja digitado o nome FIM. Imprimir a média pessoal APENAS das alunas. Imprimir também a média aritmética dos homens.

```
nome = raw_input("Digite o nome do aluno(a): ")
cont = 0
soma = 0.0

while(nome != "FIM"):
        sexo = raw_input("Digite o sexo(F/M): ").upper()
        nota1 = float(raw_input("Digite a primeira nota: "))
        nota2 = float(raw_input("Digite a segunda nota: "))
        media = (nota1 + nota2)/2.0

        if(sexo == "F"):
                print "Media da {0}: {1}".format(nome, media)
        elif(sexo == "M"):
                cont += 1
                soma += media

        nome = raw_input("Digite o nome do aluno(a): ")

if(cont != 0):
        mediaTotal = soma/cont
        print "Media dos homens da turma: {0}".format(mediaTotal)
else:
        print "Nenhum aluno homem na turma"
```

Capítulo 9

Classe String

9.1 A Ideia de Ordenação do Python e o Slice

Todas as classes/tipos do Python que possuem ideia de ordenação podem ser acessadas por partes, através de uma operação denominada slice.
Em Python cada representante de um grupo ordenado possui uma numeração na sequência, variando de 0, para o primeiro elemento e seguindo uma progressão aritmética de razão 1 até o último termo. Ou uma progressão aritmética de razão -1, e último termo -1, começando do último termo até o primeiro.

Exemplo:

P	Y	T	H	O	N
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

O acesso a partes da string se dá da seguinte forma: nomedavariavel[inicio:fim:passo]

Exemplo:

```
>>> a = 'python'
>>> a[0]
'p'
>>> a[-6]
'p'
>>> a[1:4]
'yth'
>>>a[:4]
'pyth'
>>> a[:5:2]
'pto'
>>> a[-6:-1:2]
'pto'
>>> a[-6::2]
'pto'
```

9.1.1 O Comando len(x)

Esse comando se refere a todos as classes que possuem ideia de tamanho. Seu valor de entrada x é uma variável pertencente a uma dessas classes e o comando retorna seu "tamanho".

```
>>> a = 'oi'
>>> len(a)
2
>>> a[:len(a)]
'oi'
```

9.2 Operadores

+ → Indica "Concatenação"

```
>>> a = 'estudo'
>>> b = ' dos '
>>> c = 'computadores'
>>> d = a + b + c
>>> d
'estudo dos computadores'
>>> b = 'dos'
>>> d = a + b + c
'estudosdoscomputadores'
```

∗ → Representa Repetição

```
>>> a * 3
'oioioi'
```

9.3 A Função ord(x)

A função ord(x), recebe uma string de um único caractere e retorna seu número na tabela ASCII.

```
>>> ord(a')
97
>>> ord('1')
49
```

9.4 A Função chr(x)

A função chr(x) faz o inverso da função ord. Essa função recebe um número inteiro e retorna o caractere que ele representa na tabela ASCII.

```
>>> chr(49)
'1'
>>> chr(97)
'a'
```

9.5 Métodos da Classe String

Por hora trataremos métodos como tipos diferenciados de funções, contudo mais adiante será melhor explicado a diferença entre métodos e os comandos normais. Os métodos dessa classe, assim como os das demais classes primitivas da linguagem, estão integrados (built-in) e por esse motivo não é necessário a importação de módulos para sua utilização. Ou seja, o programador pode simplesmente chamá-los e executá-los sem problemas, desde que sejam obedecidos os tipos dos argumentos de entrada e o uso dos parênteses (mesmo que não possua argumentos de entrada).

Serão apresentadas somente as principais funções da classe, e todas elas exemplificadas para melhor entendimento do leitor.

9.5.1 count

Esse método é utilizado para contar quantas vezes uma determinada string "b"aparece em uma string "a". Esse método também aceita os argumentos opcionais inicio e fim, que são utilizados a fim de restringir o tamanho da pesquisa dentro da string.
Por default inicio é a posição 0 da string e fim é a posição do ultimo caractere.
a.count (b, [inicio, fim]) → a – String, b – String, inicio – Inteiro, fim – inteiro

```
>>> a = 'python'
>>> a.count('h')
1
>>> a.count('h', 0, 2)
0
```

9.5.2 lower

Esse método é utilizado a fim de retornar uma cópia de uma string em caixa baixa.
a.lower() → a - String

```
>>> a = 'PYTHON'
>>> a.lower()
'python'
```

9.5.3 upper

Esse método é utilizado a fim de retornar uma cópia de uma string em caixa alta.
a.upper() → a – String

```
>>> a = 'python'
>>> a.upper()
'PYTHON'
```

9.5.4 replace

Esse método retorna uma nova string que substitui na string "a"todas as coincidências de uma string "b"por uma nova string "c". Também recebe um argumento de entrada opcional que chamaremos de quantidade (qtd) que representa a quantidade de substituições que serão feitas na string.
Por default qtd é toda a string.
a.replace(b, c [,qtd]) → a – String, b – String, c – String, qtd – Inteiro

```
>>> a = 'python'
>>> a.replace('p', 'c')
'cython'
```

9.5.5 split

Esse método separa a string "a"toda vez que for encontrada a string "b". Cada fracionamento da string será transformado em um item de uma lista. Esse método também recebe como dado de entrada opcional uma quantidade de vezes que será realizado o fracionamento.
Por default "b"é considerado como ' ' e qtd é considerado como toda a string.
a.split(b[,qtd]) → a – String, b – String, qtd – Inteiro

```
>>> a = 'python'
>>> a.split('y')
['p', 'thon']
```

9.5.6 isalnum

Esse método é utilizado para verificação dos tipos de caracteres de uma string. Caso a string analisada possua somente caracteres alfanuméricos retorna True, caso contrário retorna False.
a.isalnum() → a – String

```
>>> 'python'.isalnum()
True
```

9.5.7 isalpha

Esse método é utilizado para verificação dos tipos de caracteres de uma string. Caso a string analisada possua somente caracteres alfabéticos retorna True, caso contrário retorna False.
a.isalpha() → a – String

```
>>> 'python'.isalpha()
True
```

9.5.8 isdigit

Esse método é utilizado para verificação dos tipos de caracteres de uma string. Caso a string analisada possua somente caracteres númericos retorna True, caso contrário retorna False.
a.isdigit() → a – String

```
>>> 'python'.isdigit()
False
```

9.5.9 islower

Esse método é utilizado para verificação dos tipos de caracteres de uma string. Caso a string analisada possua somente caracteres que não sejam caixa alta retorna True, caso contrário retorna False.
a.islower() → a – String

```
>>> 'python'.islower()
True
```

9.5.10 isupper

Esse método é utilizado para verificação dos tipos de caracteres de uma string. Caso a string analisada possua somente caracteres que não sejam em caixa baixa retorna True, caso contrário retorna False.
a.isupper() → a – String

```
>>> 'python'.isupper()
False
```

9.6 Formatações Avançadas de Strings

9.6.1 Pular Linhas

Para mudar de linha coloca-se o parâmetro \n

```
>>> a = raw_input('\n\n digite o valor de a')


digite o valor de a
```

9.6.2 Tabulação Horizontal

Para executar uma tabulação horizontal utiliza-se o parâmetro \t. Esse comando funciona somente ao ser executado no modo programado.
O programa

```
nome = 'Antonio'
print '\t' + str(nome)
```

Terá como resultado

```
>>>
Antonio
```

9.6.3 String Unicode

Para declarar que uma string está no padrão Unicode, basta colocar a letra u antes da string.

```
>>> texto = u'string unicode'
>>> texto
u'string unicode
```

9.6.4 String Crua

A letra r antes da string indica que ela é uma string crua, ou seja, as strings de barra invertida não serão interpretadas.

```
>>> texto = r'\n\n nao vai pular linha '
>>> texto
'\\n\\n nao vai pular linha '
>>> texto = r'vou pular linha usando a \
assim posso escrever do jeito que eu quiser, contudo a \
será exibida como integrante do texto na impressão.'
>>> print(texto)
vou pular linha usando a \
assim posso escrever do jeito que eu quiser, contudo a \
será exibida como integrante do texto na impressão.
```

9.6.5 Aspas ou Apóstrofos Dentro da String

Para utilizar aspas dentro de uma string devemos utilizar o parâmetro \", o mesmo acontece com o apostrofo, porém seu parâmetro é o \'

```
>>> texto = "\"usando\""
>>> texto
'"usando aspas"'
>>> texto = '\'usando apostrofos\''
>>> texto
"'usando apostrofos'"
>>> texto = '"usando aspas"'
>>> texto
'"usando aspas"'
>>> texto = "'usando apostrofos'"
>>> texto
"'usando apostrofos'"
```

9.6.6 O Método format

Esse método é utilizado para se formatar strings. A string na qual esse método é chamado pode conter um texto literal ou campos de substituição delimitados por chaves {}. Cada campo de substituição contém ou o index numérico de um argumento, ou uma palavra chave

para um argumento. Retorna uma cópia da string onde cada campo de substituição é trocado pelo valor da string que corresponde ao seu argumento.
Exemplo:

```
>>> a = 10.5
>>> b = 4.2
>>> print 'a = {0} b = {1}'.format(a,b)
a = 10.5 b = 4.2
>>> print 'a = {:5.2f}'.format(a)
a = 10.50
```

Acessando argumentos pela posição:

```
>>> '{0}, {1}, {2}'.format('a','b','c')
'a, b, c'
>>> # alterando a ordem dos argumentos
>>> '{2}, {1}, {0}'.format('a','b','c')
'c, b, a'
>>> # repetindo argumentos
>>> '{0}, {1}, {0}'.format('a','b','c')
'a, b, a'
```

Acessando pela nome:

```
>>> print 'Meu nome é {nome}'.format(nome = 'Rojas')
Meu nome é Rojas
```

Alinhando o texto:

```
>>> print '{:<30}'.format('alinhamento esquerda')# 30 espaços
alinhamento esquerda
>>> print '{:>30}'.format('alinhamento direito')
alinhameto direita
>>> print '{:^30}'.format('centro')
centro
>>> print '{:*^30}'.format('centro')
************centro************
```

9.6.7 O Método rjust

O método rjust(x) pode ser utilizado para formatação da saída, uma vez que este define o espaçamento que será dado à direita de um caractere.
Observe o exemplo:

```
>>> for x in range (1, 5):
print str(x).rjut(2), str(x).rjust(2), str(x).rjust(2)
```

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

Esse método pode ser chamado de forma implícita em tratamentos de declarações posteriores como podemos perceber no exemplo abaixo:

```
>>> for x in range (1, 5):
print '%3d,%3d,%3d' %(x, x, x)

1,  1,  1
2,  2,  2
3,  3,  3
4,  4,  4
```

9.6.8 O Método ljust

O método ljust(x) pode ser utilizado para formatação da saída, uma vez que este define o espaçamento que será dado à esquerda de um caractere.

```
>>> for x in range (1, 5):
print str(x).ljust(2), str(x).ljust(3), str(x).ljust(4)

1  1   1
2  2   2
3  3   3
4  4   4
```

Ao contrário da rjust(x), a ljust(x) pode ser utilizada em declarações posteriores.

9.6.9 O Método center

O contrario do rjust e ljust esse método é um pouco mais complicado de usar. Para utiliza-lo deve-se passar como parâmetro obrigatório o tamanho do bloco que ele deverá ser centralizado e como parâmetro opcional com qual caractere será feito o espaçamento.
Observe:

```
>>> 'teste'.center(40)
'                 teste                  '
>>> 'teste'.center(40, '_')
'_________________teste__________________'
```

9.7 Exercícios Resolvidos

1. Escreva um programa em Python que conte a quantidade de espaços em branco de uma string.

```
frase = raw_input('String: ')
totalBranco = frase.count(' ')

print "Cont: {0}".format(totalBranco)
```

Versão sem o método count:

```
totalBranco = 0
frase = raw_input('String: ')
for i in range(0,len(frase):
        if frase[i] == ' ':
                totalBranco += 1

print "Cont: {0}".format(totalBranco)
```

2. Escreva um programa em Python que leia o nome e o imprima quantas vezes forem o número de caracteres.

```
nome = raw_input('Nome: ')

for i in range(0, len(nome)):
        print nome
```

3. Qual o resultado das seguintes execuções sabendo que:

```
>>> a = "python"
```

a) a[0] = j

```
Traceback (most recent call last):
        File "<pyshell#1>", line 1, in <module>
          a[0] = j
NameError: name 'j' is not defined
```

b) a[0] = 'j'

```
Traceback (most recent call last):
        File "<pyshell#2>", line 1, in <module>
          a[0] = 'j'
TypeError: 'str' object does not support item assignment
```

c) upper(a)

```
Traceback (most recent call last):
        File "<pyshell#3>", line 1, in <module>
          upper(a)
NameError: name 'upper' is not defined
```

d) a.isdigit()

```
False
```

e) a.lower(a.upper())

```
Traceback (most recent call last):
        File "<pyshell#4>", line 1, in <module>
          a.lower(a.upper())
TypeError: lower() takes no arguments (1 given)
```

4. Elias, um aluno que adora álgebra, decidiu que iria criptografar uma mensagem seguindo o seguinte padrão:
Se o código na tabela ascii do caractere for par: somar um ao código do caractere.
Se o código for impar multiplico por 5 e subtraio 3.
Decidido a criptografar a seguinte mensagem 'python' escreveu seu código, contudo não obteve sucesso. Caso seja necessário escreva um programa em Python e descubra o problema.

```
nome = raw_input('String: ')
nomeAux = ""
cont = 0

for i in range(0,len(nome)):
        if ord(nome[i]) % 2 == 0:
                nomeAux = nomeAux + chr(ord(nome[i]) + 1)
        else:
                nomeAux = nomeAux + chr((ord(nome[i]) * 5) - 3)
print nomeAux
```

O problema é que ao se multiplicar o código ascii de um caractere por 5, esse número pode ultrapassar o limite ascii (256), assim ocasionando o erro abaixo:

```
Traceback (most recent call last):
        File "<pyshell#5>", line 18, in <module>
          nomeAux = nomeAux + chr((ord(nome[i]) * 5) - 3)
ValueError: chr() arg not in range(256)
```

Uma modificação que solucionaria o problema seria eliminar a multiplicação e manter a subtração, como no trecho abaixo:

```
nomeAux = nomeAux + chr(ord(nome[i]) - 3)
```

5. Escreva um programa em Python que simule o método upper e lower.

Upper:

```
a = raw_input("Entre com uma string: ")
b = ''

for i in a:
        if(ord(i) > 96) and (ord(i) < 123):
                b += chr(ord(i) - 32)
        else:
                b += i
print(b)
```

Lower:

```
a = raw_input("Entre com uma string: ")
b = ''

for i in a:
        if(ord(i) > 64) and (ord(i) < 91):
                b += chr(ord(i) + 32)
        else:
                b += i
print(b)
```

6. Escreva um programa em Python que leia um número e imprima a si mesmo, o seu quadrado e o seu cubo. Seu programa deverá mostrar os resultados duas vezes uma utilizando o método rjust, e outra com o center.

```
a = float(raw_input("Entre com um número: "))
print str(a).rjust(6), str(a*a).rjust(6), str(a*a*a).rjust(6)
#utilizando a função ljust
#espaçamento 6 entre eles
print str(a).center(6), str(a*a).center(6), str(a*a*a).center(6)
#utilizando center
#espaçamento 6 entre eles (3 para cada lado)
```

7. Escreva um programa em Python que leia uma palavra e exiba seu primeiro e seu último caractere.

```
a = raw_input("Entre com uma palavra: ")
print a[0], a[len(a) - 1]
```

8. Escreva um programa em Python que leia uma frase e exiba o número de vogais e consoantes que aparecem nela.

```
nome = raw_input("Entre com um nome: ")
nomeTeste = nome.lower()
consoante = 0
vogal = 0
```

```
especiais = 0

for i in nomeTeste:
        if(i == 'a') or (i == 'e') or (i == 'i') or (i == 'o') or\
        (i == 'u'):
                vogal += 1
        elif(ord(i) < 123) and (ord(i) > 96):
                consoante += 1
        else:
                especiais += 1

print 'número de vogais: {0}\nnúmero de consoantes: {1}\nnúmero de\
caracteres especiais: {2}'.format(vogal, consoante, especiais)
```

9. Palíndromos são palavras que são idênticas se lidas da esquerda para a direita ou da direita para a esquerda, a exemplo da palavra ovo. Escreva um programa em Python que verifique se uma palavra é um palíndromo.

```
palavra = raw_input("Entre com uma palavra: ")
palindromo = True

for i in range(0, len(palavra)//2):
        if(palavra[i] != palavra[len(palavra) - i - 1]):
                palindromo = False
                break

if(palindromo):
        print "Eh um palindromo"
else:
        print "Nao eh um palindromo"
```

10. Escreva um programa em Python que leia uma cadeia e imprima a soma dos valores ASCII dos caracteres.

```
cadeia = raw_input("Entre com uma palavra: ")
soma = 0

for i in cadeia:
        soma += ord(i)

print soma
```

Capítulo 10

Módulos

Módulos são arquivos contendo o código Python que podem ser incorporados a novos programas, a fim de evitar ter de reescrever o material a ser rodado pelo interpretador e facilitar a manutenção do programa. Existem vários módulos adicionais em Python que fornecem funções e métodos que aumentam a capacidade da linguagem.

10.1 Importação de Módulos

Para incorporar as funções e métodos presentes em algum módulo ao programa deve-se importar o módulo desejado através dos comandos from e import.
A importação pode ser feita das seguintes maneiras:

10.1.1 import nome_do_modulo

Nesse caso, toda vez que desejar usar um item do módulo o programador deverá chamá-la através do seguinte comando:
nome_do_modulo.nome_do_item(argumentos)
Exemplo:

```
>>> import math
>>> math.sin(0)
0.0
```

10.1.2 from nome_do_modulo import item1, item2, ..., itemn

Nesse caso, podemos chamar os itens especificados a qualquer momento no programa como se estes fossem internos ao Python. Para casos onde é vantajoso importarmos um módulo inteiro podemos utilizar no lugar dos itens o asterisco (*).
Observe os exemplos:

```
>>> from math import sin, cos
>>> sin(0)
0.0
>>> cos(0)
```

```
1.0
>>> from math import *
>>> sin(0)
0.0
>>> tan(0)
0.0
```

10.2 O Módulo math

No decorrer dessa seção serão dados exemplos considerando o modelo de importação mais fácil, porém menos apropriado pois podem ocorrer conflitos nos nomes dos itens em alguns módulos.

```
>>> from math import *
```

10.2.1 Funções Trigonométricas Básicas do Módulo math

Arco Cosseno
Função: Calcula o arco cosseno do argumento x.
acos(x) → x é um arco medido em radianos.

```
>>> acos(1)
0.0
```

Arco Seno
Função: Calcula o arco seno do argumento x.
asin(x) → x é um arco medido em radianos.

```
>>> asin(1)
1.5707963267948966
```

Arco Tagente
Função: Calcula o arco tangente do argumento x.
atan(x) → x é um arco medido em radianos.

```
>>> atan(1)
0.7853981633974483
```

Seno
Função: Calcula o seno do argumento x.
sin(x) → x é um arco medido em radianos.

```
>>> sin(1)
0.8414709848078965
```

Cosseno
Função: Calcula o cosseno do argumento x.
cos(x) → x é um arco medido em radianos.

```
>>> cos(1)
0.5403023058681398
```

Tangente
Função: Calcula a tangente do argumento x.
tan(x) → x é um arco medido em radianos.

```
>>> tan(1)
1.5574077246549023
```

10.2.2 Outras Funções Importantes do Módulo math

Exp
Função: Calcula e**x.
exp(x).

```
>>> exp(1)
2.718281828459045
```

Log
Função: Calcula o log de x na base y.
log(x,y) → Onde x é o numero e y é a base.

```
>>> log(1, 10)
0.0
```

Fatorial
Função: Calcula o fatorial de um número inteiro.
factorial(x) → x é um inteiro.

```
>>> factorial(5)
120
```

Raiz Quadrada
Função: Retorna a raiz quadrada de um número inteiro ou real.
sqrt(x) → x é um número.

```
>>> sqrt(4)
2.0
```

Inteiro Absoluto
Função: Retorna o valor inteiro absoluto de x.
abs(x) → Onde x é um número inteiro.

```
>>> abs(-1)
1
```

Float Absoluto
Função: Retorna o float absoluto de x.
fabs(x) → onde x é um número float.

```
>>> fabs(-1.0)
1.0
```

Inteiro Maior
Função: Retorna o menor número inteiro maior que x.
ceil(x) → x é um float.

```
>>> ceil(1.9)
2.0
```

10.3 O Módulo random

Assim como o módulo math, o módulo random necessita de importação para ser utilizado. Ele contém muitas funções úteis para se trabalhar com números aleatórios.

```
>>> import random
```

10.3.1 Algumas Funções Úteis do Módulo random

Inteiro Randômico
Função: Retorna um número inteiro aleatório contido no intervalo especificado. O intervalo é fechado.
randint(a, b) → a valor inicial e b valor final: inteiros.

```
>>> random.randint(1, 10)
8
```

Intervalo Randômico
Função: Similar a randint, porém o intervalo é aberto e pode-se escolher o passo da formação da lista de números.
randrange(inicio, fim, passo) → Inicio, fim e passo: inteiros.

```
>>> random.randrange(1, 10, 3)
7
```

Shuffle
Função: Embaralha uma lista.
shuffle(self, x, random=None, int=<type 'int'>).

```
>>> a = range(1, 100)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
```

```
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99]
>>> random.shuffle(a)
>>> a
[7, 85, 12, 70, 79, 92, 96, 30, 65, 21, 34, 15, 97, 83, 41, 3, 37, 62, 56,
25, 72, 84, 73, 81, 24, 5, 23, 36, 50, 42, 44, 78, 69, 95, 54, 43, 47, 59,
71, 1, 53, 27, 38, 60, 32, 16, 99, 45, 8, 51, 98, 87, 89, 33, 14, 52, 49,
80, 11, 20, 2, 63, 57, 66, 74, 76, 18, 26, 55, 67, 86, 39, 58, 77, 75, 82,
61, 10, 6, 29, 68, 17, 48, 90, 4, 28, 35, 31, 94, 88, 40, 9, 19, 13, 64,
93, 91, 46, 22]
```

10.4 O Módulo os

O módulo **os** serve para lidar com o sistema operacional, aqui será apresentada apenas uma parte de seu conteúdo e espera-se que o programador tenha bom senso em sua utilização. Para usá-lo é necessário importá-lo.

```
>>> import os
```

10.4.1 O Método abort

O método abort é utilizado para forçar a parada do interpretador Python.
abort().

10.4.2 O Método chdir

Muda o diretório de trabalho para outro caminho especificado.
chdir(b) → Onde "b"representa uma cadeia que contém um diretório.

10.4.3 O Método chmod

Esse método altera as permissões de um arquivo
chmod(b,a) → onde "b"representa o diretório e "a"a permissão a ser atribuída.

10.4.4 O Método getcwd

Esse método retorna o diretório de trabalho atual.
getcwd().

10.4.5 O Método mkdir

Cria um diretório
mkdir(a,b) → onde "a"é um diretório e "b"o modo (por padrão 0777).

10.4.6 O Método remove

Remove um arquivo.
remove(a) → onde "a"é um diretório.

10.4.7 O Método urandom

Retorna um valor aleatório de "a" bytes para ser utilizado em criptografia. O tipo do valor retornado é uma string.
urandom(a) → onde a é o número de bytes.

10.5 Criando e Executando um Módulo

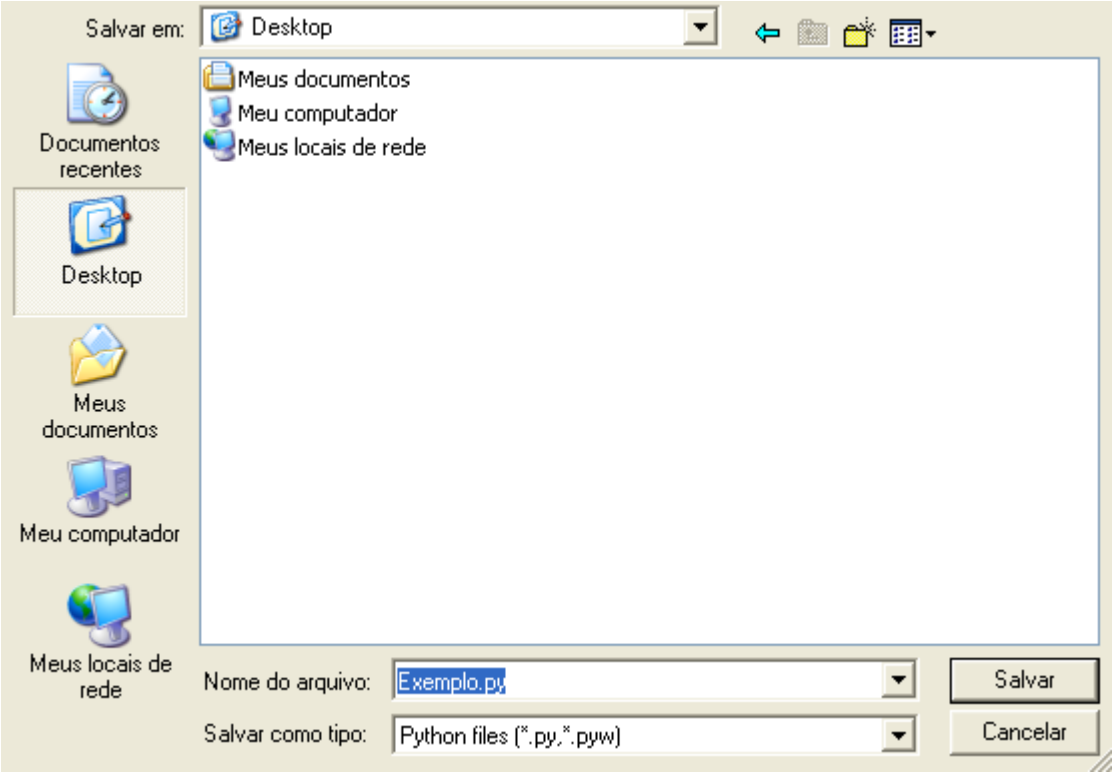
Antes de executar um módulo devemos criá-lo e salva-lo. Observe que o módulo deverá ser salvo com a extensão .py , que como o programador já deve ter percebido, torna o arquivo reconhecível pelo interpretador de Python e evita a perda do esquema de cores.

Os métodos de importação seguem os mesmos padrões especificados anteriormente no livro com o módulo math, tornando-se desnecessária a exemplificação. Observe um passo a passo simples de utilização de módulos:

```
a = 'Variável'
b = 2012
```

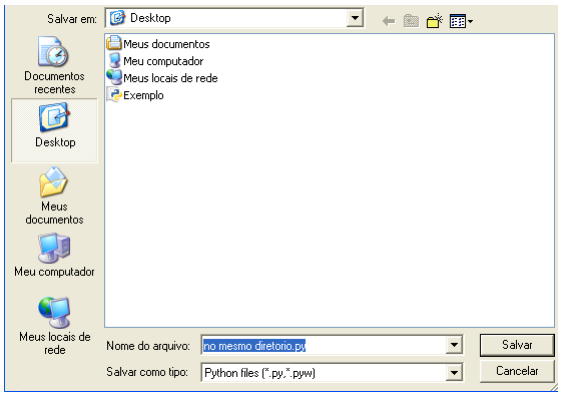
Um aviso importante é que os caracteres do módulo devem ser somente ASCII, dessa forma o módulo acima retornaria erro ao ser chamado. A forma correta é:

```
a = 'Variavel'
b = 2012
```



Observe aqui um pequeno passo a passo para sua utilização:

```
import Exemplo
print Exemplo.a
print Exemplo.b
print 'Estou no código fonte'
print 'Posso escrever o que quiser aqui'
```



Para iniciantes, o modo mais fácil de evitar problemas é salvar o código fonte no mesmo diretório do módulo. A execução do código fonte geraria o seguinte output:

```
>>>
Variavel
```

```
2012
Estou no código fonte
Posso escrever o que quiser aqui
```

10.6 Exercícios Resolvidos

1. Escreva um programa em Python que calcule o fator gama para uma velocidade v digitada pelo usuário.
   Lembrando: $\gamma = \frac{1}{\sqrt{1-(\frac{v}{c})^2}}$ e $c = 2,997 \times 10^8 m/s$

   ```
   import math

   v = float(raw_input("Digite uma velocidade v: "))
   c = 2.997 * math.pow(10, 8)
   gamma = 1/math.sqrt(1 - math.pow(v/c, 2))

   print "Fator gama: {0}".format(gamma)
   ```

2. Escreva um programa em Python que leia os coeficientes A, B e C de uma equação do segundo grau e diga o valor de suas raízes. Além disso, informe se as raízes são reais.

   ```
   import math

   a = float(raw_input("Entre com o coeficiente do termo de grau 2: "))
   b = float(raw_input("Entre com o coeficiente do termo de grau 1: "))
   c = float(raw_input("Entre com o termo independente: "))

   delta = (math.pow(b,2) - 4*a*c)

   if delta >= 0:
           x1 = (-b + math.sqrt(delta))/(2.0*a)
           x2 = (-b - math.sqrt(delta))/(2.0*a)
           print("Suas raizes sao: {0} e {1}").format(x1, x2)
   else:
           print("Raizes imaginarias")
   ```

3. Escreva um programa em Python que gere um número aleatório entre 100 e 200. O usuário do programa deve descobrir o número através de tentativa e erro. O programa irá informar se o número sorteado é maior ou menor do que o usuário informou. No final, imprimir o número de tentativas necessárias para descobrir o número.

   ```
   import random

   num = random.randint(100, 200)
   cont = 0
   tent = int(raw_input("Digite um numero aleatorio entre 100 e 200: "))
   ```

```
cont += 1

while(tent != num):
        if(tent < num):
                print "O numero sorteado eh maior"
        else:
                print "O numero sorteado eh menor"

        tent = int(raw_input("Digite um numero aleatorio entre 100 e 200: "))
        cont += 1

print "Acertou"
print "Numero de tentativas necessarias: {0}".format(cont)
```

Capítulo 11

Tratamento de Erro

O tratamento de erro (ou tratamento de exceção) é um mecanismo para tratar ocorrências que alteram o fluxo normal de execução do programa.

11.1 Tipos de Erro

11.1.1 Erros de Sintaxe

Também conhecidos como erros de parse.

```
>>> if a != True
SyntaxError: invalid syntax
```

11.1.2 Exceções

São os erros que são sintaticamente corretos, mas impossíveis de serem executados por algum motivo.

```
>>> a

Traceback (most recent call last):
        File "<pyshell#4>", line 1, in <module>
          a
NameError: name 'a' is not defined
```

11.2 Tratamento de Exceções

A seguir são apresentados os comandos que a linguagem Python fornece para tratar exceções.

11.2.1 O Comando try

Esse comando, assim como a clausula if funciona uma única vez e, diferentemente dela, precisa de uma cláusula que a torne composta, no caso o comando except.

Modelo de declaração do try:
try:
    comando ou bloco de comandos que serão verificados

11.2.2 O Comando except

Esse comando é semelhante ao else. Assim como o else só é executado quando o if retorna False, o except só é executado quando o try retorna um erro.

Diferentemente do else, o except aceita argumentos que deverão especificar os tipos de erros tratados. Caso os erros não sejam especificados na clausula eles serão entregues a um try mais externo. Caso não exista um tratador é mostrada uma mensagem de erro.

Modelo de declaração do except:
except(tipo de erro, ..., tipo de erro):
    Comando ou bloco de comandos a serem executados

```
>>> try:
        x = float(raw_input('entre com um número '))
except (valueError):
        print('Parando o programa por tipo errado')

entre com um número dez
Parando o programa por tipo errado
```

A Máscara de Exceções

O Comando except pode executar um conjunto de comandos padrão para qualquer tipo de erro que ocorra, para isso basta não apresentar argumentos.

```
>>> try:
        x = float(raw_input('entre com um número '))
except:
        print('Parando o programa por tipo errado')

entre com um número dez
Parando o programa por tipo errado
```

11.3 Provocando Exceções

O comando raise permite ao programador forçar a ocorrência de um determinado tipo de exceção.
Modelo sem o raise:

```
>>> try:
        x = float(raw_input('entre com um número '))
except (ValueError):
        print('Parando o programa por tipo errado')
```

```
entre com um número 10
```

Modelo com o raise:

```
>>> try:
        x = float(raw_input('entre com um número '))
        raise ValueError
except (ValueError):
        print('Parando o programa por tipo errado')

entre com um número 10
Parando o programa por tipo errado
```

11.4 O Comando finally

O comando finally é utilizado para ações de limpeza, sendo executado tanto quando ocorrer e também quando não ocorrer exceções.
Veja os exemplos:

```
>>> try:
        raise ValueError
except (TypeError):
        print('Não vai passar por aqui')
finally:
        print('vou ser executado de qualquer forma')

vou ser executado de qualquer forma

Traceback (most recent call last):
        File "<pyhell#56>", line 2, in <module>
          raise VaueError
ValueError

>>> try:
        raise ValueError
except (ValueError):
        print('Vai passar por aqui')
finally:
        print('vou ser executado de qualquer forma')

Vai passar por aqui
vou ser executado de qualquer forma
```

11.5 Exercícios Resolvidos

1. Escreva um programa em Python para ler um número através de raw_input e o converta para float.

```
while True:
        try:
                a = float(raw_input("Entre com o valor: "))
                print a
                break
        except:
                print "Ocorreu um problema com sua entrada, insira\
                novamente os dados"
```

2. Escreva um programa em Python que leia dois valores e execute a divisão entre eles.

```
while True:
        try:
                a = float(raw_input("Entre com um valor: "))
                b = float(raw_input("Entre com outro valor: "))
                break
        except:
                print "Ocorreu um problema com sua entrada, insira\
                novamente os dados"

try:
        print "Seu numero eh {0}".format(a/float(b))
except:
        print "Ocorreu um erro na divisao"
```

3. Em uma fábrica existe um botão de desligamento forçado. Quando este botão é ativado ele envia o valor 11 para o programa interno. Escreva um programa em Python que pare assim que receber o valor 11.

```
while True:
        try:
                a = float(input("Entre com um valor: "))

                if(a == 11):
                        raise(ValueError)

        except(ValueError):
                print "Desligando as maquinas"
                break
```

4. Escreva o mesmo programa do exercício 1 deste capítulo sem o uso do comando try.

```
a = raw_input("Entre com um numero: ")
cont = 0
num = False

while(num == False):
        if(a.isdigit()):
                print float(a)
                num = True
        else:
                cont = 0

                for i in a:
                        if i == '.':
                                cont += 1
                        elif i.isdigit():
                                pass
                        else:
                                cont += 2
                                break

        if cont > 1:
                print "Nao eh um numero"
        else:
                print a
                num = True

        if(num == False):
                a = raw_input("Entre com um numero: ")
```

11.6 Para Saber Mais

Refaça os exercícios dos capítulos anteriores tratando os erros.

Capítulo 12

A Classe list

Lista é um conjunto ordenado de valores identificado por um índice.
Uma lista ou *list* em Inglês possui as seguintes características:

- Os elementos PODEM ser alterados.*
- Os elementos PODEM ser repetidos.
- Os itens da lista são separados por vírgula (,) e escritos entre colchetes [].
- Uma lista PODE conter diferentes tipos de dados inclusive outras listas. Uma lista dentro de outra lista denomina-se aninhada.
- Os valores armazenados em uma lista podem ser acessados usando colchetes [] e [:] cujos índices iniciam em 0 quando contado a partir do inicio da lista e -1 o último elemento quando contado a partir do fim da lista.
- Quando os dados forem do tipo string, um operador de slice pode ser aplicado.
- Listas possuem métodos que podem ser aplicados a elas. Um método é semelhante a uma função, mas são invocados de forma diferente: objeto.método(argumentos).
  Por exemplo: para adicionar um valor a uma lista usa-se o método append. Assim temos: b.append(5) que significa inclua o valor 5 na lista b.

* Listas são tipos básicos mutáveis do Python, ou seja, podem ser adicionados novos elementos, ou ainda alterá-los. Porém, vale ressaltar que a alteração é feita diretamente na memória causando dessa forma alteração em todos os objetos que apontam para a mesma lista.
O construtor dessa classe é chamado através do comando list (x).

12.1 Métodos da Classe list

12.1.1 O Método append

Esse método é utilizado para acrescentar um objeto **b** como integrante no final da lista **a**.
a.append(b) → a - Lista, b - Qualquer

```
>>> b = [1, 2, 3, 4]
>>> b.append('objeto')
>>> b
[1, 2, 3, 4, 'objeto']
```

12.1.2 O Método count

Esse método conta quantas vezes o objeto **b** aparece na lista **a**, ao contrário das strings esse método não recebe argumentos opcionais de entrada.
a.count(b) → a - Lista, b - Qualquer

```
>>> a = [1, 1, 1, 1, 2, 3]
>>> a.count(1)
4
```

12.1.3 O Método insert

O método insert é muito similar ao método append, uma vez que ambos adicionam um objeto como integrante na lista, contudo esse método recebe como argumento, além do dado que será que será inserido na lista, a posição que ele será inserido.
Caso a posição seja maior que a lista, o item será acrescentado ao final.
a.insert(pos, b) → a - Lista, b - Qualquer

```
>>> a = [1,2,3,4,5,6,8]
>>> a.insert(6,7)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8]
```

12.1.4 O Método pop

O método pop é utilizado para a remoção de um item da lista. Recebe como argumento de entrada a posição que este item ocupa.
a.pop(pos) → a – Lista, pos – Inteiro

```
>>> a = [1,2,3,4]
>>> a.pop(3)
4
>>> a
[1, 2, 3]
```

12.1.5 O Método remove

O método remove, remove a primeira ocorrência de um determinado objeto em uma lista.
a.remove(b) → a – Lista, b – Qualquer

```
>>> a = [1,1,1,1,1]
>>> a.remove(1)
>>> a
[1, 1, 1, 1]
```

12.1.6 O Método sort

O método sort é o oposto do shuffle do módulo random, sendo utilizado para ordenar listas.
a.sort() → a – Lista

```
>>> a = [1,3,4,2,6,5]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5, 6]
```

12.1.7 O Método reverse

Esse método é utilizado para inverter a ordem que os itens de uma lista estão posicionados.
a.reverse() → a - Lista

```
>>> a = [1,2,3,4,5,6]
>>> a.reverse()
>>> a
[6, 5, 4, 3, 2, 1]
```

12.1.8 Tratando listas como vetores e matrizes

Definição 12.1 Sejam m e n números naturais. Uma matriz $m \times n$ (m por n) é um quadro formado por $m \cdot n$ elementos de **mesma natureza** dispostos em m linhas e n colunas.

Representamos por

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

uma matriz $A = [a_{ij}]_{m \times n}$, em que o índice i, $i = 1, 2, 3, \cdots, m$, indica a linha e o índice j, $j = 1, 2, 3, \cdots, n$, indica a coluna.

Exemplo 12.1

$$\begin{bmatrix} 2 & -1 & 4 & 0 \\ 0 & 3 & -7 & 23 \end{bmatrix}$$

é uma matriz 2×4.

Exemplo 12.2

$$\begin{bmatrix} 0 & 1 & 12 \\ 5 & 8 & -9 \\ 9 & 0 & 4 \end{bmatrix}$$

é uma matriz 3×3.

Definição 12.2 Matriz linha ou vetor é toda matriz do tipo $1 \times n$.

Python não possui o conceito de matriz. Para tanto usa-se listas para representar os vetores e listas de listas para representar as matrizes. É importante destacar que uma lista é um objeto *inteirável*, isto é, pode-se percorrer todos os elementos da lista com um comando **FOR**.

Exemplo 12.3 Escreva um programa em Python que leia o vetor A com 5 valores reais. Exiba os valores na ordem em que foram lidos.

```
A=[] # cria a variavel A do tipo lista
for i in range(0,5):
x=float(raw_input('Digite um valor '))
A.append(x) #adiciona a A o valor lido
print"Valores lidos"
for j in A: #percorre a lista
print "{:^10.2f}".format(j) #centraliza o valor com 2 decimais
```

Exemplo 12.4 Agora trataremos de uma matriz de duas dimensões.
Observe no exemplo a seguir as etapas comentadas para criação da matriz.
Escreva um programa Python que leia uma matriz de duas dimensões com 3x3 elementos inteiros.

```
# passo 1 -  cria uma lista de  3 valores
a=[None]*3
for i in range(3):
# passo 2 para cada elemento da lista cria outra lista de 3 valores
a[i]=[None]*3
# seu formato será a=[[a11,a12,a13],[a21,a22,a23],[a31,a32,a33]]
# passo 3 ler todos os elementos da lista
for i in range(3): # para cada linha
for j in range(3):
#le os elementos das colunas
#[i] é a referencia da linha e [j] da coluna
a[i][j]=int(raw_input('Digite um numero '))
```

A *interface* humano-computador é muito importante mesmo em programa simples. Vamos melhorar a legibilidade da entrada de dados alterando a última linha do programa:

```
a[i][j]=int(raw_input('Digite A ['+str(i+1)+','+str(j+1)+'] --> '))
```

Este comando produzirá a seguinte tela:
A impressão da matriz deve ser realizada sob a forma de tabela, permitindo ao leitor identificar claramente as linhas e colunas.

```
print'\n{:*^30}\n'.format('Matriz Lida') # cabeçalho
for i in range(3): # para cada linha
for j in range (3):
# imprima os elementos das colunas e continua imprimindo
print ('{:>10.2f}').format(a[i][j]),
print #muda de linha
```

Este trecho de programa produzirá a seguinte saída:

```
Digite A [1,1] --> 1
Digite A [1,2] --> 2
Digite A [1,3] --> 3
Digite A [2,1] --> 4
Digite A [2,2] --> 5
Digite A [2,3] --> 6
Digite A [3,1] --> 7
Digite A [3,2] --> 8
Digite A [3,3] --> 9

*********Matriz Lida**********

1.00    2.00    3.00
4.00    5.00    6.00
7.00    8.00    9.00
```

12.2 Exercícios Resolvidos

1. Escreva um programa em Python que leia o vetor A com 5 valores inteiros. Determine um vetor com a seguinte lei de formação: Os termos de ordem impar de A são multiplicados por 3 Os termos de ordem par de A são multiplicados por 2.

```
a = []
v = []

for i in range(0, 5):
        val = int(raw_input("Entre com um valor: "))
        a.append(val)

        if((val % 2) == 0):
                v.append(val * 2)
        else:
                v.append(val * 3)

print "Vetor A: {0}\nVetor V: {1}".format(a, v)
```

2. Escreva um programa em Python que sorteie 6 números da loteria, que consiste de valores de 1 até 500, e armazene em uma lista.

```
import random

v = []
val = random.randint(1, 500)

while len(v) != 6:
        v.append(val)
        val = random.randint(1, 500)

        while val in v:
```

```
                val = random.randint(1, 500)

print v
```

3. Escreva um programa em Python que leia duas matrizes $m \times n$ e calcule sua soma.

```
M = int(raw_input("Entre com o numero de linhas: "))
N = int(raw_input("Entre com o numero de colunas: "))

m1 = [None] * M
m2 = [None] * M
m3 = [None] * M

print "Matriz A"
for i in range(0, M):
        m1[i] = [None] * N

        for j in range(0, N):
                m1[i][j] = float(raw_input("Entre com o elemento a{0}{1}: "\
                .format(i + 1,j+1)))

print "Matriz B"
for i in range(0, M):
        m2[i] = [None] * N
        m3[i] = [None] * N

        for j in range(0, N):
                m2[i][j] = float(raw_input("Entre com o elemento b{0}{1}: "\
                .format(i + 1,j+1)))
                m3[i][j] = m1[i][j] + m2[i][j]

print "Soma das matrizes A e B:"

for i in m3:
        print i
```

4. Escreva um programa em Python que leia um número indeterminado de nomes e insira-os em uma lista até que se deixe o nome em branco. Imprimir o maior nome da lista.

```
nomes=[] # cria a lista nomes

while True:
        nome = raw_input('Digite um nome ')
        if nome == '':
                break
        nomes.append(nome)
```

```
#percorrendo a lista
#determinando o maior
t = 0
mnome = ''
print '\n{:*^30}\n'.format('maior nome')

for i in nomes:
        print i,
        print len(i)
        l = len(i)

        if l > t:
                t = l
                mnome = i

print 'O maior nome eh {:20s}'.format(mnome)
```

5. Escreva um programa em Python que leia e faça a multiplicação de duas matrizes de qualquer ordem.

```
numLinA = int(raw_input("Entre com o numero de linhas da Matriz A: "))
numColA = int(raw_input("Entre com o numero de colunas da Matriz A: "))

numLinB = int(raw_input("Entre com o numero de linhas da Matriz B: "))
numColB = int(raw_input("Entre com o numero de colunas da Matriz A: "))

a = [None] * numLinA
b = [None] * numLinB

print "Matriz A"

for i in range(0, numLinA):
        a[i] = [None] * numColA

        for j in range(0, numColA):
                a[i][j] = float(raw_input("Entre com o coeficiente a{0}{1}: "\
                .format(i+1,j+1)))

print "Matriz B"

for i in range(0, numLinB):
        b[i] = [None] * numColB

        for j in range (0, numColB):
                b[i][j] = float(raw_input("Entre com o coeficiente b{0}{1}: "\
```

```
                .format(i+1,j+1)))

x = [0] * numLinA
for i in range(0, numLinA):
        x[i] = [0] * numColB

for i in range(0, numLinA):
        for j in range(0, numColB):
                for k in range(0, numColA):
                        x[i][j] += a[i][k] * b[k][j]

for i in x:
        print(i)
```

6. Escreva um programa em Python que leia uma palavra e embaralhe suas letras.

```
from random import *

nome=raw_input('Nome: ')
lista=[]

for i in nome:
        lista.append(i) # cada letra eh um elemento da lista

shuffle(lista)
nomee=''

for i in lista:
        nomee+=i

print 'Nome embaralhado --> {:20s}'.format(nomee)
```

7. O Bubble Sort é um algoritmo de ordenação que consiste em percorrer uma lista tomando seus elementos um a um e comparando-os com o seguinte, e caso estejam fora de ordem troca as posições.
   Escreva um programa em Python que gere uma lista desordenada de 20 elementos e a ordene em ordem crescente com o Bubble Sort.

```
import random

a = range(1, 21)
random.shuffle(a)
print a

# ordenacao
for i in range(len(a) - 1):
```

```
        for j in range(i+1, len(a)):
                if a[i] > a[j]:
                        a[i],a[j] = a[j],a[i]

print a
```

8. O Selection Sort é um algoritmo baseado em mover sempre o menor/maior valor para a posição desejada. Após isso o processo de identificação do menor valor e a alocação na posição desejada se repete para todos os n-1 valores restantes. Escreva um programa em Python que gere uma lista desordenada de 20 elementos e a ordene em ordem crescente com o Selection Sort.

```
import random

a = range(1, 21)
random.shuffle(a)
print a

n = len(a)
for i in range(n - 1):
menor = i
        for j in range(i + 1, n):
                if(a[j] < a[menor]):
                        menor = j

        a[i], a[menor] = a[menor], a[i]

print a
```

9. A Pesquisa ou Busca Sequencial é um algoritmo para expressar um tipo de pesquisa em vetores ou listas de modo sequencial, i.e., elemento por elemento, até encontrar o elemento desejado. Escreva um programa em Python que gere uma lista de 20 elementos desordenados. Gere um elemento x aleatório e busque-o usando a Busca Sequencial.

```
import random

lista = range(1, 21)
random.shuffle(lista)

busca = random.randint(1,20)
print ('Numero a procurar: {0}'.format(busca))

achou = False
for i in range(0, 20):
        if(busca == lista[i]):
                achou = True
```

```
                break

if achou:
        print "O numero {0} esta no index {1} da lista".\
        format(busca, i)
        print lista
```

10. A Pesquisa ou Busca Binária é um algoritmo de busca em vetores que segue o paradigma de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor. Escreva um programa em Python que leia nomes inseridos pelo usuário. Leia um nome e busque-o usando a Busca Binária.

```
nome=[]
n = int(raw_input('Digite a quantidade de nomes '))

for i in range(n):
        x = raw_input('Nome: ')
        nome.append(x)

#ordena os nomes
nome.sort()
# le nome a procurar
nomep = raw_input('Nome a procurar: ')
inicio = 0
fim = len(nome) - 1
achei = False

while inicio <= fim :
        meio = (inicio + fim)/2
        if nome[meio] == nomep:
                achei = True
                break
        elif nomep > nome[meio]:
                inicio = meio + 1
        else:
                fim = meio - 1

if achei:
        print 'achei'
else:
        print 'nao achei'
```

12.3 Para Saber Mais

```
>>> help(list)
```
Existe também um módulo array, que talvez seja interessante para alguns.

```
>>> import array
>>> help(array)
```

Capítulo 13

Conjuntos

Um conjunto é um agrupamento sem noção de sequência, sem elementos repetidos e imutáveis. Sets são estruturas de dados do tipo conjunto.
Ao transformarmos qualquer tipo composto de dados em um set, eliminamos toda e qualquer repetição de integrantes. O construtor da classe set é o comando set(x). Observe o exemplo:

```
>>> a = 'quem cola nao sai da escola'
>>> set(a)
set(['a', ' ', 'c', 'e', 'd', 'i', 'm', 'l', 'o', 'n', 'q', 's', 'u'])
```

13.1 Métodos de Sets

13.1.1 add

Esse método adiciona um objeto ao set.
a.add(b) → a – set, b – objeto.

```
>>> a = set([])
>> a
set([])
>>> a.add(11)
>>> a
set([11])
```

13.1.2 copy

Esse método retorna uma cópia do set.
a.copy() → a – set.

```
>>> a = 'python'
>>> a = set(a)
>>> b = a.copy()
>>> b
set(['p', 't', 'y', 'h', 'o', 'n'])
```

13.1.3 difference

Esse método retorna a diferença entre os conjuntos.
a.difference(b) → a – set, b – set.

```
>>> a = 'python'
>>> a = set(a)
>>> b = 'pearl'
>>> b = set(b)
>>> a.difference(b)
set(['y', 'h', 't', 'o', 'n'])
```

13.1.4 discard

Esse método descarta um objeto do set.
a.discard(b) → a – set , b – objeto.

```
>>> a = 'python'
>>> a = set(a)
>>> a.discard('y')
>>> a
set(['h', 'o', 'n', 'p', 't'])
```

13.1.5 intersection

Esse método retorna um set correspondente a interseção dos dois sets apresentados.
a.intersection(b) → a – set , b – set.

```
>>> a = 'python'
>>> a = set(a)
>>> b = 'pearl'
>>> b = set(b)
>>> a.intersection(b)
set(['p'])
```

13.1.6 symmetric_difference

Esse método retorna a diferença simétrica entre os dois sets.
a.symmetric_difference(b) → a – set, b – set.

```
>>> a = 'python'
>>> a = set(a)
>>> b = 'pearl'
>>> b = set(b)
>>> a.symmetric_difference(b)
set(['a', 'e', 'h', 'l', 'o', 'n', 'r', 't', 'y'])
```

13.1.7 union

Esse método retorna a união entre os dois sets.
a.union(b) → a - set, b - set.

```
>>> a = 'python'
>>> a = set(a)
>>> b = 'pearl'
>>> b = set(b)
>>> a.union(b)
set(['a', 'e', 'h', 'l', 'o', 'n', 'p', 'r', 't', 'y'])
```

13.2 Comando del

O comando del pode ser utilizado para apagar slices e variáveis.

```
>>> a = 'pascal'
>>> del a
>>> a

Traceback (most recent call last):
        File "<pyshell#105>", line 1, in <module>
          a
NameError: name 'a' is not defined
>>> a = 'python
>>> a = list(a)
>>> del a[0:len(a):3]
>>> a
['y', 't', 'o', 'n']
```

13.3 Comando enumerate

Através do uso dessa comando pode-se exibir o objeto e sua respectiva posição ao percorrer um objeto sequencial qualquer.

```
>>> for i,j in enumerate(['testando','um','possibilidade']):
        print i,j

0 testando
1 uma
2 possibilidade
```

13.4 Comando zip

O comando zip é utilizado para percorrer simultaneamente duas ou mais sequências. Observe o exemplo:

```
>>> for i,j,k in zip(range(1,5), range(1,20,2), range(5,10)):
        print ('%2d,%2d,%2d') % (i,j,k)

 1, 1, 5
 2, 3, 6
 3, 5, 7
 4, 7, 8
```

13.5 Comando reversed

O comando reversed é usada para percorrer uma sequencia em ordem reversa. Observe o exemplo:

```
>>> for i in reversed('alo'):
        print i

o
l
a
```

13.6 Comando sorted

O comando sorted ordena uma lista sem uma sequência sem alterá-la na memória, diferentemente do que ocorre no método sort de listas. Observe:

```
>>> for i in sorted('arqueiro'):
        print i

a
e
i
o
q
r
r
u
```

13.7 Exercícios Resolvidos

1. Escreva um programa em Python que gere uma lista de 200 números aleatórios entre 1 e 50 e imprima: Uma lista ordenada e sem repetições dos números, a quantidade de números que foram retirados da lista inicial e a lista inicial, nessa ordem. Importante: Você não deverá utilizar mais de duas variáveis.

```
import random

a = []

while len(a) < 200:
        a.append(random.randint(1, 50))

print "Lista ordenada e sem repeticoes:\n{0}\n".\
format(sorted(list(set(a))))
print "Quantidade de numeros retirados: {0}\n".\
format(len(a) - len(sorted(list(set(a)))))
print "Lista inicial:\n{0}\n".format(a)
```

2. Escreva um programa em Python que gere uma lista desordenada de tamanho 30 e a imprima em ordem decrescente. Após isso imprima a lista desordenada inicial. Seu programa deverá utilizar somente uma variável.

```
import random

a = range(1, 31)
random.shuffle(a)

print "Lista em ordem decrescente:"
for i in reversed(sorted(a)):
        print i,

print "Lista inicial:\n{0}".format(a)
```

3. Escreva um programa em Python que gere duas listas de tamanhos aleatórios e imprima cada uma das duas listas e os termos que pertencem às duas simultaneamente. O tamanho das listas não deve superar 100.

```
import random

a = range(0, random.randint(1, 101))
b = range(0, random.randint(1, 101))

random.shuffle(a)
random.shuffle(b)

print "Lista A:\n{0}\n".format(a)
print "Lista B:\n{0}\n".format(b)
print "Tamanho de A: {0}\nTamanho de B: {1}\n".format(len(a), len(b))
print "Interseccao de A com B:\n{0}".format(set(a).\
intersection(set(b)))
```

4. Escreva um programa em Python que gere duas listas de tamanhos aleatórios e imprima cada uma das duas listas e os termos que pertencem a uma mas não pertencem a outra. O tamanho das listas não deve superar 100.

```
import random

a = range(0, random.randint(1, 101))
b = range(0, random.randint(1, 101))

random.shuffle(a)
random.shuffle(b)

print "Lista A:\n{0}\n".format(a)
print "Lista B:\n{0}\n".format(b)
print "Tamanho de A: {0}\nTamanho de B: {1}\n".format(len(a), len(b))
print "Termos unicos de A:\n{0}\n".format(set(a).difference(set(b)))
print "Termos unicos de B:\n{0}\n".format(set(b).difference(set(a)))
```

5. Escreva um programa em Python que gere duas listas de tamanhos aleatórios e imprima cada uma das duas listas e a diferença simétrica entre elas. O tamanho das listas não deve superar 100.

```
import random

a = range(0, random.randint(1, 101))
b = range(0, random.randint(1, 101))

random.shuffle(a)
random.shuffle(b)

print "Lista A:\n{0}\n".format(a)
print "Lista B:\n{0}\n".format(b)
print "Tamanho de A: {0}\nTamanho de B: {1}\n".format(len(a), len(b))
print "Diferenca simetrica de A com B:\n{0}\n".format(list(set(b).\
symmetric_difference(set(a))))
```

Capítulo 14

Outras Classes de Python

14.1 A Classe Tupla

Uma tupla é outro tipo de estrutura de dados similar a Lista.

Uma tupla consiste de uma série de valores separados por vírgula. Ao contrário das listas, no entanto, as tuplas são representadas entre parênteses.

As principais diferenças entre listas e tuplas são: Listas são incluídas em colchetes ([]), e os seus elementos e tamanho pode ser alterado, enquanto tuplas são colocados entre parênteses (()) e seus elementos não podem ser modificados. Tuplas podem ser tratadas como listas de somente leitura.

O construtor de tuplas é chamando a partir do comando tupple(x).

14.2 Dicionários

Dicionários são objetos que podem ser modificados, com características sequenciais não ordenadas e que são utilizados a fim de implementar mapeamentos.

Cada objeto do dicionário é relacionado com uma chave.

Essa classe é declarada entre chaves e utiliza-se dois pontos ":"para separar objetos de suas respectivas chaves.

O construtor dessa classe é chamado através do comando dict(x)

14.3 Métodos da Classe dict

14.3.1 copy

Esse método é utilizado para retornar uma cópia do dicionário.
a.copy () → a – dicionário.

```
>>> a = {'algo': 'relacionado'}
>>> b = a.copy()
>>> b
{'algo': 'relaciondo'}
```

14.3.2 keys

O método keys retorna uma lista contendo cada uma das chaves de dicionário.
a.keys() → a – dicionário.

```
>>> a = {'algo': 'relacionado'}
>>> a.keys()
['algo']
```

14.3.3 clear

Esse método remove todos os itens do dicionário
a.clear () → a – dicionário.

```
>>> a = {'algo': 'relacionado'}
>>> a.clear()
>>> a
{}
```

14.3.4 items

Esse método retorna uma lista contendo cada um dos itens pertencentes ao dicionário.
a.items() → a – dicionário.

```
>>> a = {'algo': 'relacionado'}
>>> a.items()
[('algo', 'relacionado')]
```

14.3.5 update

O método update insere as chaves e itens de **b** em **a**.
a.update (b) → a – dicionário, b – dicionário.

```
>>> a = {'algo: 'relacionado'}
>>> b = {'outra': 'coisa'}
>>> a.update(b)
>>> a
{'algo': 'relaciondo', 'outra': 'coisa'}
```

14.3.6 iteritems

Com esse método torna-se possível percorrer um dicionário e exibir sua chave simultaneamente.

```
>>> d = {'nom': 'Joao', 'sexo' : 'Masculino', 'idade':20}
>>> for i,j in d.iteritems():
        print i,j
```

```
idade 20
sexo Masculino
nome Joao
```

14.4 Exercícios Resolvidos

1. Escreva um programa em Python que transforme uma lista em uma tupla.

   ```
   import random

   a = range(1, 20)
   random.shuffle(a)
   a = tuple(a)
   print a
   ```

2. Escreva um programa em Python que transforme uma lista em dicionário.

   ```
   import random

   a = [[1,2],[3,4],[5,6]]
   a = dict(a)
   print a
   ```

3. Escreva um programa em Python que faça a concatenação (merge) de duas tuplas de 20 valores cada uma.

   ```
   import random

   a = range(0, 20)
   b = range(0, 20)

   random.shuffle(a)
   random.shuffle(b)

   a = tuple(a)
   b = tuple(b)
   a += b
   print a
   ```

4. Escreva um programa em Python que imprima duas strings como objetos de uma tupla, objetos de uma lista e como a primeira sendo chave de dicionário da segunda.

   ```
   a = raw_input("Entre com a primeira string: ")
   b = raw_input("Entre com a segunda string: ")
   lista = []
   ```

```
dici = {}

lista.append(a)
lista.append(b)
tupla = tuple(lista)
dici[lista[0]] = lista[1]

print lista
print tupla
print dici
```

5. Com base em seus conhecimentos sobre a linguagem Python, diga qual será o resultado dos seguintes comandos:

   (a)
   ```
   >>> a = 1
   >>> b = (2)
   >>> c = b + a
   >>> c
   3
   ```

   (b)
   ```
   >>> a = 1
   >>> b = 2,
   >>> c = b + a

   Traceback (most recent call last):
           File "<pyshell#18>", line 1, in <module>
             c = b + a
   TypeError: can only concatenate tuple (not "int") to tuple
   ```

   (c)
   ```
   >>> a = 1,
   >>> b = 2,
   >>> a + b
   (1, 2)
   ```

   (d)
   ```
   >>> a = 1,
   >>> b = 2,
   >>> a - b

   Traceback (most recent call last):
           File "<pyshell#28>", line 1, in <module>
             a - b
   TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'
   ```

   (e)
   ```
   >>> a = 1,
   >>> b = 2,
   >>> a * b

   Traceback (most recent call last):
           File "<pyhell#32>", line 1, in <module>
   ```

```
          a *b
TypeError: can't multiply sequence by non-int of type 'tuple'
```

(f)
```
>>> tuple('Python')
('P', 'y', 't', 'h', 'o', 'n')
```

(g)
```
>>> dict(range(0, 20))

Traceback (most recent call last):
        File "<pyshell#38>", line 1, in <module>
          dict(range(0,20))
TypeError: cannot convert dictionary update sequence element #0to a sequence
```

(h)
```
>>> dici = {}
>>> dici["oi"] = "tchau"
>>> dici['oi']
'tchau'
```

(i)
```
>>> dici = {}
>>> dic["oi"] = "tchau/t"
>>> dici['oi']
'tchau/t'
```

14.5 Para Saber Mais

Para saber mais sobre tuplas e dicionários

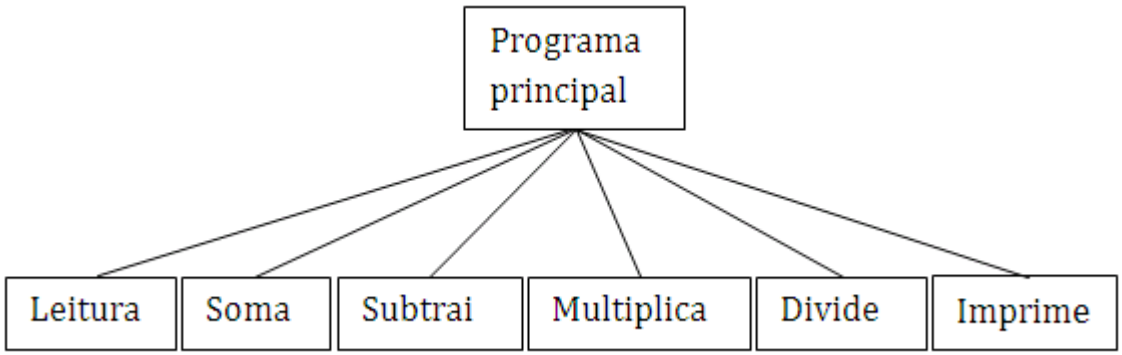
```
>>> help(tuple)
>>> help(dict)
```

Capítulo 15

Funções

Para facilitar a criação, manutenção e especialmente a correção dos programas, ao invés de criarmos uma sequência única de comandos um após o outro podemos dividir o programa em sub-rotinas ou funções. As funções podem ser definidas como trechos de código com uma função específica.

Mais especificamente, uma função é uma porção do código que pode receber argumentos de entrada e saída, realizando procedimentos e retornando ou não um valor. Observe abaixo o esquema resumido do funcionamento de um programa através de funções:



As vantagens de se dividir um programa em funções são as seguintes:

- Simplificação do código - torna o programa mais legível e organizado.
- Reutilização do código - ao invés de reescrever o mesmo código várias vezes, utilizamos uma função para executar o mesmo código várias vezes.
- Facilidade de manutenção - A localização e correção de um erro fica mais fácil

15.1 O Conceito de Escopo

Por definição, refere-se à região do código onde as variáveis são diretamente acessíveis. Durante qualquer execução existem escopos internos, onde são encontradas as variáveis locais, o escopo intermediário contendo as variáveis globais e o escopo externo contendo variáveis predefinidas.

15.2 O Conceito de Parâmetro

Os parâmetros têm como finalidade servir como ponto de comunicação bidirecional entre a função e o programa principal, ou com uma sub-rotina hierarquicamente de nível mais alto. Os parâmetros podem ser formais ou reais.

Serão parâmetros formais quando forem declarados como argumentos de entrada de uma função, e serão tratados no escopo local da mesma forma que seriam no escopo global.

Serão parâmetros reais quando forem passados na chamada da função no escopo global.

Caso o leitor esteja meio confuso sobre esse assunto, não se desespere, pois este será exemplificado com comentários na área de criação de funções.

Existem dois tipos mais conhecidos de passagem de parâmetro que são: valor e referência.

15.2.1 Passagem de Parâmetro por Valor

A passagem por valor caracteriza-se pela não alteração do valor do parâmetro real quando o parâmetro formal é manipulado dentro da sub-rotina.

Em Python as variáveis escalares são passadas dessa forma.

15.2.2 Passagem de Parâmetro por Referência

A passagem de parâmetro por referência caracteriza-se pela manipulação do parâmetro real quando o parâmetro formal é manipulado dentro da sub-rotina.

A alteração efetuada no parâmetro formal corresponde à alteração no parâmetro real.

Em Python não existe a passagem por referência, contudo ao tratarmos de objetos do tipo mutável, por razões lógicas sobre seu tipo e funcionamento obtemos um resultado parecido com esse tipo de passagem. Observe o seguinte código:

```
def referencia (a):
        b = len(a) + 1
        a.append(b)
a = [1,2,3,4,5,6,7]
referencia(a)
print a
```

Como esperado o resultado será:

```
>>>
[1, 2, 3, 4, 5, 6, 7, 8]
>>>
```

15.3 O Comando return

O comando return (x) é um comando que finaliza a função e retorna como valor dela o valor da variável x.

15.4 Criando uma Função

def nome da função (argumento1,argumento2,...,argumento n):
    Comando ou bloco de comandos
    return(valor à ser retornado) # caso a função retorne algo
Continuação do código

Exemplo:

```
def produto (a,b): #a e b são parâmetros formais
        '''função simples para retornar o produto de dois termos'''
#caso não se lembre das docstrings volte ao início do livro
        c = c * d
        return c #retornando o valor do produto
e = float(raw_input("Entre com o primeiro valor"))
f = float(raw_input("Entre com o segundo valor"))
d = produto(e,f) #e e f são parâmetros reais
print "sua soma é", d
```

15.5 O Conceito de Variável Global e Variável Local

Quando uma atribuição é feita dentro de uma função, por padrão o Python criará uma variável dentro do escopo local da função, desse modo não será possível acessar a variável de fora da dela. Após o termino da execução da função as variáveis serão destruídas.

As variáveis globais são aquelas que podem ser acessadas em qualquer área, ou seja, aquelas que não estão dentro de um escopo local. Dessa forma exemplos de variáveis globais são todas as variáveis criadas até o presente momento.

Caso o leitor ainda assim não entenda segue um exemplo especificando as variáveis locais e globais.

```
def soma (a,b):
        '''função simples para retornar a soma de dois termos'''
#caso não se lembre das docstrings volte ao início do livro
        c = a + b #a, b, c são variáveis locais
        return c #retornando o valor da soma
e = float(raw_input("Entre com o primeiro valor"))
f = float(raw_input("Entre com o segundo valor"))
d = soma(e,f) #a = e, b = f valor retornado atribuido à variável d
print "sua soma é", d
```

15.6 O Comando Global

Mesmo dentro de uma função pode-se transformar uma variável local em global. Para realizarmos tal tarefa utilizamos o comando global **x**, que fará com que todas as variáveis **x** abaixo dessa chamada sejam tratadas como globais.
Exemplo:

```
def dentro():
        global a
        a = 5
def fora():
        a = 11
        print 'O valor de a é', a
        dentro()
        print 'O a local mudou para', a
fora()
print 'O a global mudou para', a

>>>
O valor de a é 11
O a local mudou para 11
o a global mudou para 5
```

Como podemos notar o comando global não alterou o valor dentro da função **fora**, permanecendo o valor 11, contudo dentro do escopo global a variável a passou a existir com o valor 5, conforme foi definido na função **dentro**.

Em casos simples, como o apresentado, o problema pode ser facilmente evitado da forma apresentada a seguir:

```
def dentro(x):
        global a
        a = 5
        return a
def fora():
        a = 11
        print 'O valor de a é', a
        print 'O a local mudou para', dentro(a)
fora()
print 'O a global mudou para', a
```

15.7 O Comando type

O comando type(x) retorna o tipo da variável x (int, float, complex, str,...), permitindo assim que o programador verifique o tipo do argumento de entrada da função antes de passá-lo e executá-la.

```
>>> a = 11
>>> type(a)
<type 'int'>
>>> type(a) == int
True
```

15.8 Atribuição Default

Em alguns casos, a entrada de dados do usuário é algo opcional, isso se dá porque o programador pode definir um valor padrão para a entrada de argumentos que será passado para a função caso não possua dados para a entrada.

def nome (argumento1=valor default 1, ... , argumento n = valor default n):
    comando ou bloco de comandos

Exemplo:

```
def teste(a = 't', b = 'e', c = 's', d = 't', e = 'e'):
        print(a, b, c, d, e)
x = 1000
teste()
teste(x)
teste(b = x, d = 1)

>>>
('t', 'e', 's', 't', 'e')
(1000, 'e', 's', 't', 'e')
('t', 1000, 's', 1, 'e')
```

15.9 Passando um Número Variável de Argumentos

A passagem de um número indefinido de argumentos sempre deverá aparecer após os argumentos conhecidos, caso a função possua.

Declara-se a passagem de um número indefinido de entradas de argumentos de uma função através do uso do símbolo *.
Observe o exemplo:

```
def teste (*a):
        print(a)
teste (1,2,3,4,5,6,7,8,9)

>>>
(1, 2, 3, 4, 5, 6, 7, 8, 9)
>>>
```

15.10 Comandos do Python que Recebem Funções como Argumentos

15.10.1 filter

Esse comando recebe como argumentos uma função e uma sequência. O comando aplica os elementos da sequência na função e retorna uma outra sequencia apenas com os elementos

que a função retornou verdadeiro. Os tipos retornados em casos de string e tupla não serão modificados, nos demais serão transformados em tipo lista.

```
def impar(x):
        return x%2 != 0
b = filter(impar, range(1,10))
print (b)

>>>
[1, 3, 5, 7, 9]
>>>
```

15.10.2 map

Esse comando, assim como o filter, recebe como parâmetros de entrada uma função e uma sequência. Esse comando aplica na função cada item da sequencia e retorna a lista de valores retornados.

```
def quadrado(x):
        return x**2
b = map(quadrado, range(1,10))
print (b)

>>>
[1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

15.10.3 reduce

O comando reduce aplica, recursivamente, uma operação a uma sequência de números e é chamada da seguinte forma:
reduce(função, lista ou tupla).
Observe:

```
>>> reduce(list.__add__, [[1, 2, 3], [4, 5, 6], [7, 8, 9]], [])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

15.11 Recursividade

Funções recursivas são funções que chamam a si mesmas de forma que, para resolver um problema maior, utilizam a recursão para chegar as unidades básicas do problema em questão e então calcular o resultado final. Para melhor compreensão, usaremos como exemplo a sequência de Fibonacci. Nela nós temos dois casos base: O primeiro elemento da sequencia sempre é 0, assim como o segundo elemento da sequência também é sempre 1.

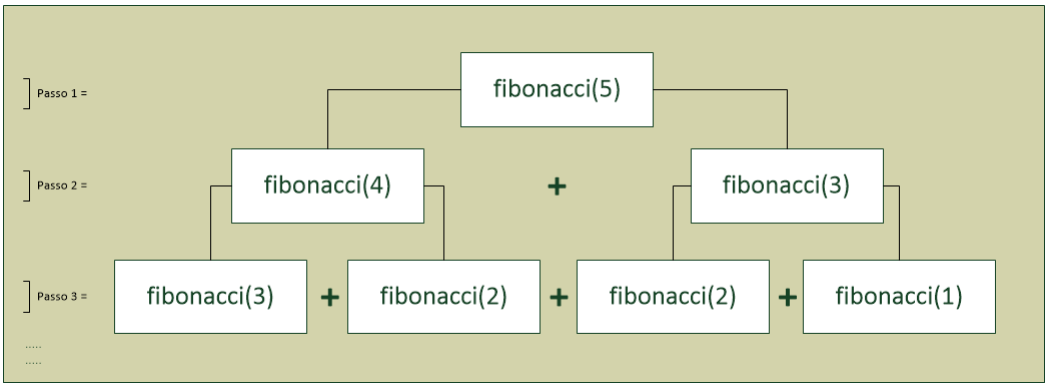
Temos então o caso base. Para calcular os demais, utilizamos o seguinte parâmetro: o valor "i"da sequência será o valor "i-1"da sequência somado ao valor "i-2"da sequência. Ou

seja: fibonacci(i) = fibonacci(i - 1) + fibonacci(i - 2). Sendo assim, podemos recorrer à própria função para calcular o valor de um elemento da sequencia. Vejamos como isto se traduz em código Python:

```
def fibonacci(num):
        if num < 2:
                return num
        else:
                return fibonacci(num - 1) + fibonacci(num - 2)
```

Dessa forma, a função se torna muito mais simples de entender e legível. Porém, funções recursivas também tem seus problemas.

O primeiro problema está na repetição do cálculo do mesmo valor. Vejamos um exemplo: calcular o quinto valor da sequencia.



Perceba que, no passo 2 precisamos calcular o fibonacci(4) e o fibonacci(3). Então, no passo 3, precisamos calcular novamente o fibonacci(3) para encontrar o valor do fibonacci(4) do passo 2.

Outro problema é que a pilha de chamadas não pode ser infinita, e alguns compiladores ou interpretadores limita o número máximo de chamadas a funções na pilha. Exemplificando: Quando chamamos fibonacci(5), esta chamada irá chamar primeiramente fibonacci(4), sendo assim, a primeira chamada é colocada numa pilha e o controle passa para a segunda chamada. Então, fibonacci(4) chama fibonacci(3), sendo colocada também na pilha enquanto fibonacci(3) assume o controle. Agora imagine que você chamou fibonacci(1000). Quando a chamada chegar a fibonacci(2) serão 998 chamadas de função na pilha, sem contar que cada chamada ainda irá chamar uma segunda ramificação.

15.11.1 Exemplo

Fatorial:

```
def fat(num):
        if num == 0:
                return 1

        return num * fat(num - 1)
```

15.12 Exercícios Resolvidos

1. Escreva qual será a saída de cada um dos seguintes comandos:

   (a)
```
import random
def impares(lista):
        for i in lista:
                if i % 2 == 0:
                        lista.remove(i)
a = range(1, 20)
random.suffle(a)
print impares(a)

None
>>> #como podemos ver já que a função não retorna
>>> #nada o interpretador mostra um objeto vazio
```

   (b)
```
import random
def impares(lista):
        lista2 = []
        for i in lista:
                if i % 2 != 0:
                        lista2.append(i)
        return lista2
a = range(1, 20)
random.shuffle(a)
print sorted(impares(a))

>>>
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>>
```

   (c)
```
import random
def pop(lista):
        lista.pop(2)
a = range(1, 20)
random.shuffle(a)
pop(a)
print len(a)

>>>
18 #um número aleatório entre 1 e 19
>>>
```

2. Escreva um programa em Python que calcule a soma de dois números. Seu programa deverá conter a função calcular.

```
def calcular(a,b):
        c = a + b
```

```
        return c

a = float(raw_input("Entre com um numero: "))
b = float(raw_input("Entre com outro numero: "))
print calcular(a,b)
```

3. Uma sequencia é dita feia quando possui números que possuem somente os fatores 2 ou 3.
   Observe:
   2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 64, 72, 81, 96, 108,...
   Escreva um programa em Python que contenha as funções de cálculo e impressão da sequencia.
   Importante você não deve imprimir o objeto do tipo lista!

```
def calcula():
        atual = 2
        contador = 0
        pot3 = 3
        lista = []

        while len(lista) < 1000:
                if atual < pot3:
                        lista.append(atual)
                        atual = lista[contador] * 2
                        contador += 1
                else:
                        lista.append(pot3)
                        pot3 *= 3

        return lista

def imprime(lista):
        for i in lista:
                print i

imprime(calcula())
```

4. Durante nossos primeiros contatos com a matemática, fomos ensinados uma técnica facilitadora na soma, o chamado "vai um". Escreva um programa em Python que leia dois números inteiros e imprima o valor da soma e a quantidade de "vai um" utilizada.

```
def swap(a,b):
        try:
                if a < b:
                        a,b = b,a
        except:
```

```
                print "Erro nos numeros"

        return a,b

def convert(a,b):
        a,b = str(a),str(b)
        return a,b

def igualatam(a,b):
        aumento = ""

        while len(b) + len(aumento) != len(a):
                aumento += '0'

        aumento += b
        return aumento

def calcula(a,b):
        c = ""
        d = ""
        contavai = 0
        vai = 0
        tam = len(a) - 1

        while tam > -1:
                conta = int(a[tam]) + int(b[tam]) + vai

        if conta > 9:
                vai = 1
                contavai += 1
                conta -= 10
        else:
                vai = 0

        c += str(conta)
        tam -= 1

        if vai != 0:
                c += str(vai)

        for i in reversed(c):
                d += i

        return d, contavai

a,b = int(raw_input("Entre com o par ordenado para a soma "))
```

```
a,b = swap(a,b)
a,b = convert(a,b)
b = igualatam(a,b)
c,d = calcula(a,b)

print "Valor da soma: {0}".format(c)
print "Numero de vai 1: {0}".format(d)
```

5. Escreva um programa em Python para imprimir:
   1
   2 2
   3 3 3
   .....
   n n n n n n ... n
   para um n informado pelo usuário. Use uma função que receba um valor n inteiro e imprima até a n-ésima linha.

```
def imprimir(n):
        for i in range(1, n + 1):
                for j in range(1, i + 1):
                        print i,

                print

n = int(raw_input('Entre com um n: '))
imprimir(n)
```

6. Escreva um programa em Python com uma função que necessite de um argumento. A função retorna o valor de caractere 'P', se seu argumento for positivo, e 'N', se seu argumento for zero ou negativo.

```
def calcula(n):
        if(n <= 0):
                return 'N'
        else:
                return 'P'

n = float(raw_input('Entre com um numero: '))
print calcula(n)
```

7. Escreva um programa em Python com uma função chamada somaImposto. A função possui dois parâmetros formais: taxaImposto, que é a quantia de imposto sobre vendas expressa em porcentagem e custo, que é o custo de um item antes do imposto. A função "altera" o valor de custo para incluir o imposto sobre vendas.

```
def somaImposto(taxaImposto,custo):
        return (0.01*taxaImposto)*custo + custo
```

```
custo = float(raw_input("Entre com o custo do item: "))
taxaImposto = float(raw_input('Entre com a taxa de imposto sobre esse item: '))
somaImposto = somaImposto(taxaImposto, custo)

print "Valor do item apos imposto: {0}".format(somaImposto)
```

8. Escreva um programa em Python com uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -¿ 721.

```
def inverte(string):
        invertido = ""

        for i in range(0, len(string)):
                invertido += string[len(string) - i - 1]

        return invertido

string = raw_input("Entre com um numero: ")
print inverte(string)
```

9. Escreva um programa em Python que converta da notação de 24 horas para a notação de 12 horas. Por exemplo, o programa deve converter 14:25 em 2:25 P.M. A entrada é dada em dois inteiros. Deve haver pelo menos duas funções: uma para fazer a conversão e uma para a saída. Registre a informação A.M./P.M.. Inclua um loop que permita que o usuário repita esse cálculo para novos valores de entrada todas as vezes que desejar.

```
def converte(h, m):
        hora = ''

        if(h == '12'):
                hora = h
                turno = ' P.M.'
        elif(h == '00' or h == '24'):
                hora = '12'
                turno = ' A.M.'
        elif(int(h) > 12):
                hora = str(int(h) - 12)
                turno = ' P.M.'
        else:
                hora = h
                turno = ' A.M.'

        hora += ':' + m + turno

        return hora
```

```
def imprimir(cont):
        print cont

while True:
        print('Deixe em branco para sair')

        h = raw_input('Informe a hora: ')
        if h == "":
                break

        m = raw_input('Informe os minutos: ')
        imprimir(converte(h, m))
```

10. Escreva um programa em Python que use a função valorPagamento para determinar o valor a ser pago por uma prestação de uma conta. O programa deverá solicitar ao usuário o valor da prestação e o número de dias em atraso e passar estes valores para a função valorPagamento, que calculará o valor a ser pago e devolverá este valor ao programa que a chamou. O programa deverá então exibir o valor a ser pago na tela. Após a execução o programa deverá voltar a pedir outro valor de prestação e assim continuar até que seja informado um valor igual a zero para a prestação. Neste momento o programa deverá ser encerrado, exibindo o relatório do dia, que conterá a quantidade e o valor total de prestações pagas no dia. O cálculo do valor a ser pago é feito da seguinte forma. Para pagamentos sem atraso, cobrar o valor da prestação. Quando houver atraso, cobrar 3% de multa, mais 0,1% de juros por dia de atraso.

```
def valorPagamento(VP, NumDias):
        Vpm = VP

        if(NumDias > 0):
                Vpm *= 1.03

        valorC = round(Vpm *((1 + 0.001) ** NumDias), 2)
        listaVc.append(valorC)

        return valorC

listaVc = []
ct = 0
while True:
        VP = float(raw_input('Digite o valor da prestacao: (0 para sair) '))

        if VP == 0:
                break

        NumDias = int(raw_input('Quantos dias que esta em atraso: '))
```

```
        print 'Valor corrigido: {0}'.format(valorPagamento(VP, NumDias))
        ct += 1

print 'Quantidade de prestacoes pagas: {0}'.format(ct)
print 'Valor total de prestacoes pagas no dia: R${0}'.\
format(round(sum(listaVc), 2))
```

15.13 Exercícios de Jogos

1. Jogo da Forca (6 chances)

```
def ChecarLetra(letra, palavra, linha):
        acertou = False

        for i in range(0, len(palavra)):
                if(letra == palavra[i]):
                        linha[i] = palavra[i]
                        acertou = True

        return acertou

palavra = raw_input("Entre com a palavra a ser adivinhada: ").upper()
linha = ['_'] * len(palavra)
letrasUsadas = []
chances = 6
ganhou = False

while(True):
        print "Chances Restantes: {0}".format(chances)
        print linha
        print letrasUsadas

        letra = raw_input("Entre com uma letra: ").upper()

        if(len(letra) > 1):
                #arriscando palavra inteira
                if(letra == palavra):
                        for i in range(0, len(palavra)):
                                linha[i] = palavra[i]

                        ganhou = True
                        break
                else:
                        break
        elif(ChecarLetra(letra, palavra, linha) == False):
```

```
                chances -= 1
                letrasUsadas.append(letra)

        if(chances == 0):
                break
        elif(not('_' in linha)):
                ganhou = True
                break

if(ganhou):
        print "Parabens, voce ganhou"
        print linha
else:
        print "Que pena, voce perdeu"
        print linha
        print "A palavra era: {0}".format(palavra)
```

2. Pedra, Papel e Tesoura

```
import random

def EscolherJogada(simbolos):
        mao = raw_input("Entre com Pedra/Papel/Tesoura: ").upper()

        while(mao not in simbolos):
                print "Entrada invalida. Tente novamente."
                mao = raw_input("Entre com Pedra/Papel/Tesoura: ").upper()

        return mao

def EscolherJogComp(simbolos):
        i = random.randint(0, 2)
        mao = simbolos[i]

        print "Computador jogou {0}.".format(mao)
        return mao

def ChecarVencedor(player, comp):
        if(player == comp):
                return "EMPATE"
        elif(player == "PEDRA"):
                if(comp == "PAPEL"):
                        return comp
                else:
                        return player
        elif(player == "PAPEL"):
```

```
                if(comp == "TESOURA"):
                        return comp
                else:
                        return player
        elif(player == "TESOURA"):
                if(comp == "PEDRA"):
                        return comp
                else:
                        return player

simbolos = ['PEDRA', 'PAPEL', 'TESOURA']
jogando = True
while(jogando):
        player = EscolherJogada(simbolos)
        comp = EscolherJogComp(simbolos)

        result = ChecarVencedor(player, comp)
        while(result == "EMPATE"):
                print "Empate. Jogue novamente.".format(player)
                player = EscolherJogada(simbolos)
                comp = EscolherJogComp(simbolos)
                result = ChecarVencedor(player, comp)

        if(result == player):
                print "{0} ganha {1}. Voce venceu!".format(player, comp)
        else:
                print "{0} ganha {1}. Voce perdeu!".format(comp, player)

        resp = raw_input("Deseja jogar novamente? (S/N) ").upper()
        if(resp != "S"):
                jogando = False
```

3. Jogo da Velha

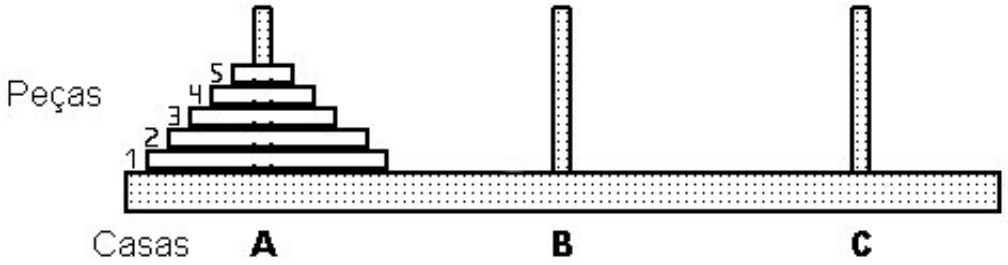
```
import sys #Modulo do sistema
import random

def CriarTabuleiro():
        tabuleiro = [''] * 3 #Cria 3 linhas
        for i in range(0, 3):
                tabuleiro[i] = [''] * 3 #Para cada linha cria 3 colunas

        return tabuleiro

def DesenharTabuleiro(tabuleiro):
        for i in range(0, 3):
```

```
                for j in range(0, 3):
                        if(j == 1):
                                #Metodo para impressao sem pular linha
                                sys.stdout.write('|')
                                sys.stdout.write(str(tabuleiro[i][j]).
                                                center(3))
                                #Imprime divisorias verticais
                                sys.stdout.write('|')
                        else:
                                sys.stdout.write(str(tabuleiro[i][j]).
                                                center(3))

                if(i < 2):
                        #Imprime divisorias horizontais
                        print '\n' + ('-' * 11)

        print '\n'

def EscolherSimbolo():
        player = ''
        comp = ''

        simbolo = raw_input("Com qual simbolo quer jogar? (X/O) ").upper()
        while player == '':
                #Checa simbolo de entrada
                if(simbolo == 'X' or simbolo == 'O'):
                        player = simbolo

                        if(player == 'X'):
                                comp = 'O'
                        else:
                                comp = 'X'
                else: #Se o simbolo for invalido, pede a entrada novamente
                        print "Entre com um simbolo valido."
                        simbolo = raw_input("Com qual simbolo quer jogar? \
(X/O) ").upper()

        return player, comp

def PreencherTabuleiro(tabuleiro, simbolo, linha, coluna):

        if(linha < 0 or linha > 2 or coluna < 0 or coluna > 2 or
                tabuleiro[linha][coluna] != ''):
                print "Jogada invalida. Tente novamente"
                return False
        else:
```

```
                tabuleiro[linha][coluna] = simbolo
                DesenharTabuleiro(tabuleiro)

        return True

def CriarJogadas(): #Gera todas as jogadas possiveis para o comp
        lista = []

        for i in range(0, 3):
                for j in range(0, 3):
                        lista.append(str(i) + ',' + str(j))

        return lista

def ChecarTabuleiro(tabuleiro, simbolo):

        if((tabuleiro[0][0] == simbolo and tabuleiro[0][1] == simbolo and
        tabuleiro[0][2] == simbolo) or (tabuleiro[1][0] == simbolo and
        tabuleiro[1][1] == simbolo and tabuleiro[1][2] == simbolo) or
        (tabuleiro[2][0] == simbolo and tabuleiro[2][1] == simbolo and
        tabuleiro[2][2] == simbolo) or (tabuleiro[0][0] == simbolo and
        tabuleiro[1][0] == simbolo and tabuleiro[2][0] == simbolo) or
        (tabuleiro[0][1] == simbolo and tabuleiro[1][1] == simbolo and
        tabuleiro[2][1] == simbolo) or (tabuleiro[0][2] == simbolo and
        tabuleiro[1][2] == simbolo and tabuleiro[2][2] == simbolo) or
        (tabuleiro[0][0] == simbolo and tabuleiro[1][1] == simbolo and
        tabuleiro[2][2] == simbolo) or (tabuleiro[0][2] == simbolo and
        tabuleiro[1][1] == simbolo and tabuleiro[2][0] == simbolo)):
                return True

        return False

#Main
tabuleiro = CriarTabuleiro()
DesenharTabuleiro(tabuleiro)

player, comp = EscolherSimbolo()

resp = raw_input("Deseja comecar? (S/N) ").upper()
if(resp == "S"):
        turno = True
else:
        turno = False

jogadas = CriarJogadas()
ganhou = False
```

```
while(True): #Loop do game

        if(turno): #Vez do Jogador
                linha = (int(raw_input("Entre com a linha: ")) - 1)
                coluna = (int(raw_input("Entre com a coluna: ")) - 1)

        jogada = PreencherTabuleiro(tabuleiro, player, linha, coluna)
        #Pede nova jogada caso espaco ja tenha sido usado
        while(not jogada):
                linha = (int(raw_input("Entre com a linha: ")) - 1)
                coluna = (int(raw_input("Entre com a coluna: ")) - 1)
                jogada = PreencherTabuleiro(tabuleiro, player, linha, coluna)
        else:
                print "Vez do computador"
                jogada = jogadas[random.randint(0, len(jogadas) - 1)]
                linha = int(jogada.split(',')[0])
                coluna = int(jogada.split(',')[1])

                PreencherTabuleiro(tabuleiro, comp, linha, coluna)

        #Retirando jogada usada das possibilidades do comp
        jogadas.remove(str(linha) + ',' + str(coluna))

        if(ChecarTabuleiro(tabuleiro, player)): #Checa se jogador venceu
                ganhou = True
                break
        elif(ChecarTabuleiro(tabuleiro, comp)): #Checa se comp venceu
                break
        elif(not jogadas): #Empate
                ganhou = "Empate"
                break

        turno = not turno

if(ganhou == "Empate"):
        print "O jogo empatou"
elif(ganhou):
        print "Parabens, voce venceu"
else:
        print "Que pena, voce perdeu"
```

4. Torre de Hanoi



Torre de Hanói é um "quebra-cabeça"que consiste em uma base contendo três pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação. O número de discos pode variar sendo que o mais simples contém apenas três.

```
def Hanoi(n, inicial, destino, temporario):
        if(n > 0):
                Hanoi(n - 1, inicial, temporario, destino)
                print 'Mova um anel do pino {0} para o pino {1}'\
                      .format(inicial,destino)
                Hanoi(n - 1, temporario,  destino, inicial)
```

Capítulo 16

Arquivos

Alguns conceitos básicos de Python relacionados a arquivos utilizados neste curso:

- Serão utilizados apenas arquivos do tipo "texto". Ou seja, serão gravados e lidos conjuntos de caracteres;
- Python admite apenas o método de acesso sequencial;
- Python não possui o conceito de "registro"para isso iremos usar o artificio de gravar como csv -*comma separated values* e depois da leitura separar os valores.

16.1 O Comando open

O comando open abre um arquivo para a utilização, ou seja, retorna um objeto de tipo arquivo. Esse comando é chamado da seguinte forma:

variavel= open (diretório_do_arquivo,modo) → diretório_do_arquivo - string, modo - string
Esta variável passa a ser a referencia do arquivo dentro do programa.

16.1.1 Os Modos que um Arquivo Pode ser Aberto

'r' → Somente para leitura, valor default de abertura de arquivos.
'w' → Somente para escrita, caso exista qualquer tipo de conteúdo o arquivo será apagado ao ser aberto desse modo.
'a' → Abrir o arquivo para adição. Qualquer escrita será adicionada ao fim do arquivo.

Podemos também abrir o arquivo com esses 3 modos em binário, basta acrescentarmos o b a um desses modos. Observe o exemplo:

```
>>> arq = open('c:\exemplo.txt, 'wb')
```

Dentro do programa todas as vezes que formos nos referir ao arquivo devemos usar arq e não exemplo.txt.

16.2 O Método close

O método close é utilizado para fechar um arquivo e liberar recursos. Vale ressaltar que qualquer tentativa de acessar o arquivo novamente resultará em falha.
Esse método normalmente não recebe argumentos e é chamado da seguinte forma:
nome_do_arquivo.close()
Observe o exemplo:

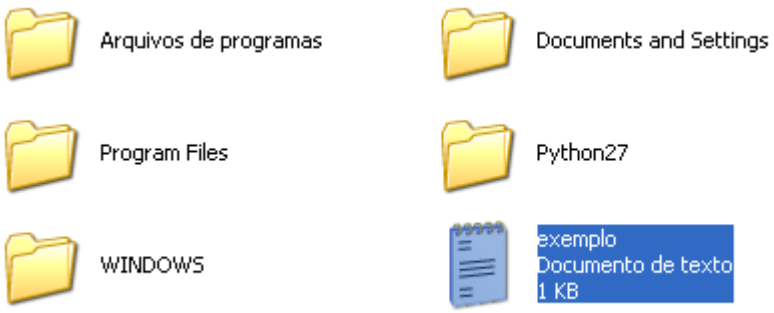
```
>>> arq = open('c:\exemplo.txt', 'w')
>>> arq.close()
```

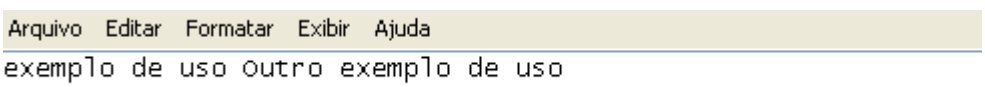
16.3 O Método write

O método write é utilizado para escrever uma string em um arquivo. É chamado da seguinte forma:
nome_do_arquivo.write(a) → a - string
Observe os exemplos:

```
>>> arq = open('c:\exemplo.txt', 'w')
>>> arq.write('exemplo de uso ')
>>> algo = 'Outro exemplo de uso'
>>> arq.write(algo)
>>> arq.close()
```

Podemos também acessar os dados escritos no arquivo através do Windows acessando o diretório e o arquivo especificado. Observe:





16.4 O Método read

O método read é utilizado para ler uma determinada quantidade de dados. Esse método recebe um argumento opcional chamado size, ou tamanho, que caso não seja especificado será considerado como o tamanho total do arquivo. Esse argumento é importante para evitar que programadores inexperientes tentem abrir arquivos cujo conteúdo é maior do que a memória disponível no computador.
Esse comando é chamado da seguinte maneira:
Nome_do_arquivo.read(tamanho) → tamanho em bytes.
Vale ressaltar inclusive que uma vez atingido o fim do arquivo (EOF), o comando retorna uma string vazia (‘’).
Observe o exemplo com base em nosso arquivo escrito anteriormente:

```
>>> arq = open('c:\exemplo.txt', 'r')
>>> arq.read()
'exemplo de uso Outro exemplo de uso'
>>> arq.read()
''
>>> arq.close()
>>>
```

16.5 O Método readline

O método readline lê uma única linha do arquivo, avaliado através do caractere de retorno de linha (\n) e que será mantido na impressão após a chamada do método. Em sua segunda chamada lê a segunda linha do arquivo, em sua terceira, a terceira linha e assim sucessivamente até encontrar o EOF. Caso seja chamado após o final do arquivo o método retornará uma string vazia.
O readline é declarado da seguinte forma:
Nome_do_arquivo.readline()
Observe o exemplo:

```
>>> arq = open('c:\exemplo.txt', 'r')
>>> arq.readline()
'exemplo de uso'
>>> arq.readline()
''
```

16.6 O Método readlines

O método readlines retorna uma lista, onde cada elemento é uma linha do arquivo. Ela recebe como parâmetro opcional o sizehint, que informa a quantidade de bytes aproximado que será lido, essa quantidade será ultrapassada em caso de necessidade de finalizar a linha.
O método readlines é declarada da seguinte forma:
Nome_do_arquivo.readlines(sizehint)

Após ser lido o EOF, caso seja chamado novamente o método retornará uma lista vazia.
Observe o exemplo:

```
>>> arq = open('c:\exemplo.txt', 'r')
>>> arq.readlines()
['exemplo de uso Outro exemplo de uso']
>>> arq.readlines()
[]
>>> arq.close()
```

16.7 O Método tell

O método tell retorna um inteiro que indica a posição corrente de leitura ou escrita no arquivo, medida em bytes.
Esse método é chamado da seguinte forma:
Nome_do_arquivo.tell()

16.8 O Método seek

O método seek é utilizado para deslocar a posição corrente de leitura ou escrita. Recebe como atributos deslocamento e de_onde.
Esse método é chamado da seguinte forma:
Nome_do_arquivo.seek(deslocamento,de_onde) → deslocamento é um inteiro medido em bytes a serem deslocados e de_onde pode assumir os seguintes valores: 0 para indicar o inicio do arquivo (valor default); 1 para indicar a posição atual e 2 para indicar o fim do arquivo.

16.9 O Método truncate

Esse método "trunca" o arquivo em determinado tamanho. O tamanho por default é a posição atual.
Independente do valor de entrada para o tamanho a posição não será alterada.

Importante: Se o valor do tamanho exceder o tamanho do arquivo atual os resultados podem variar dependendo do sistema operacional. Além disso, esse método não funcionará se o arquivo estiver aberto como somente para leitura.

16.10 Simulando Registros

Um registro é uma coletânea de dados sobre uma mesma informação. Cada informação é usualmente chamada de "campo". Por exemplo: Um registro Dados Pessoais pode-se caracterizar como sendo o conjunto de campos nome,idade,cpf,endereço.
Se Python tem apenas arquivos tipo texto pergunta-se como agrupar e desagrupar estas informações. A solução parece estar em usar arquivos tipo csv *comma separated values* onde as informações são separadas por virgula e gravadas sob a forma de tabela onde cada registro

ocupa uma linha.
Isto acarreta uma dificuldade adicional: todas as informações de um registro são separadas por virgula exceto a última que contém um caracter *carriage return* que faz a mudança de linha. A lógica adotada será incluir uma virgula entre cada campo e após o último campo incluir \n.
O exemplo apresentado a seguir irá mostrar passo-a-passo o desenvolvimento do programa.
Escreva um programa em Python que grave em um arquivo o nome e a idade de um conjunto indeterminado de pessoas. Depois leia as informações gravadas e classifique por idade. Exiba o nome e a idade de todas as pessoas.
O primeiro passo é planejar o programa indicando como será organizado o arquivo e quais módulos serão elaborados.
Para a primeira pergunta usaremos um arquivo csv;
E os módulos a serem elaborados são:

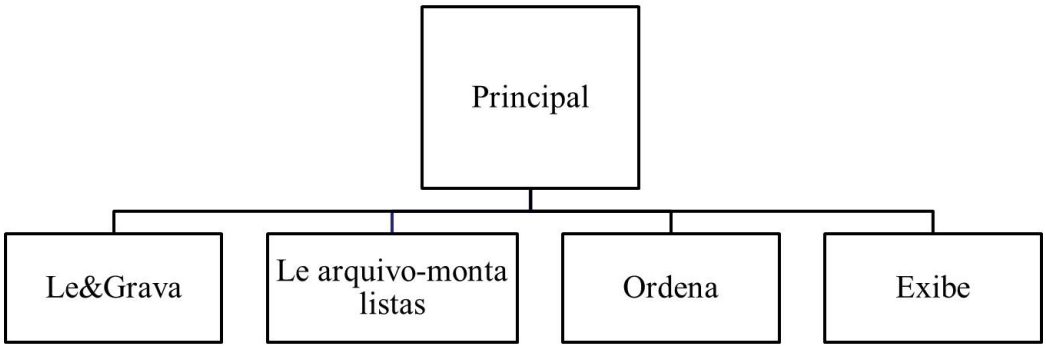


Figura 16.1: Diagrama Hierarquia de Módulos

O módulo Le&Grava tem a seguinte lógica:

Abre o arquivo para escrever
Repete até acabar
Lê do teclado informações
Monta para gravar
Grava
Retorna

Corresponde ao seguinte módulo

```
def grava():
arq=open('prog2.txt','w')
while True:
nome=raw_input('Digite um nome --> ')
```

```
idade=raw_input('Digite uma idade --> ')
arq.write(nome+','+idade+'\n')
fim=raw_input('Deseja continuar ? --> ')
if fim =='n':
break
arq.close()
return
```

O módulo Le arquivo e monta as listas tem a seguinte lógica:

Abre o arquivo para leitura
Repete até o fim do arquivo
Acessa cada linha
Separa campos
Monta listas
Retorna

Corresponde ao seguinte módulo

```
def le():
nomes=[]
idades=[]
arq=open('prog2.txt','r)
for linha in arq:
valores=linha.split(',')
nomes.append(valores[0])
idades.append(int(valores[1]))
arq.close()
return nomes,idades
```

O módulo Ordena tem a seguinte lógica:

```
def ordem(nome,idade):
n=len(nome)
for i in range(0,n-1):
for j in range(i+1,n):
if idade[i]>idade[j]:
nome[i],nome[j]=nome[j],nome[i]
idade[i],idade[j]=idade[j],idade[i]
return  nome,idade
```

O módulo Exibe tem a seguinte lógica:

```
def exibe(nome,idade,n):
print'\n{:*^30}\n'.format('Arquivo ordenado')
for j in range(n):
print'{:<30}   {:<10}'.format(nome[j],idade[j])
return
```

O programa principal tem a seguinte lógica:

```
# programa principal
grava()
nome,idade=le()
nome,idade,n=ordem(nome,idade)
exibe(nome,idade,n)
```

16.11 Exercícios Resolvidos

Aqui está um arquivo notas_estudantes.txt que contém uma linha para cada aluno de uma turma de estudantes. O nome de cada estudante está no início da cada linha e é seguido pelas suas notas.

```
jose 10 15 20 30 40
pedro 23 16 19 22
suzana 8 22 17 14 32 17 24 21 2 9 11 17
gisela 12 28 21 45 26 10
joao 14 32 25 16 89
```

1. Usando o arquivo de texto notas_estudantes.txt escreva um programa em Python que imprime o nome dos alunos que têm mais de seis notas.

```
arq = open('notas_estudantes.txt', 'r')

linha = arq.readline()
while linha:
numNotas = 0
valores = linha.split()

for i in range(1, len(valores)):
numNotas += 1

if(numNotas > 6):
print valores[0]

linha = arq.readline()

arq.close()
```

2. Usando o arquivo texto notas_estudantes.txt escreva um programa em Python que calcula a média das notas de cada estudante e imprime o nome e a média de cada estudante.

```
arq = open('notas_estudantes.txt', 'r')
```

```
linha = arq.readline()
while linha:
soma = 0
numNotas = 0
valores = linha.split()

for i in range(1, len(valores)):
soma += int(valores[i])
numNotas += 1

media = soma/float(numNotas)
print "Nome: {0} - Media: {1}".format(valores[0], media)

linha = arq.readline()

arq.close()
```

3. Usando o arquivo texto notas_estudantes.txt escreva um programa em Python que calcula a nota mínima e máxima de cada estudante e imprima o nome de cada aluno junto com a sua nota máxima e mínima.

```
arq = open('notas_estudantes.txt', 'r')

linha = arq.readline()
while linha:
notaMin = 999
notaMax = -1
valores = linha.split()

for i in range(1, len(valores)):
if(int(valores[i]) < notaMin):
notaMin = int(valores[i])

if(int(valores[i]) > notaMax):
notaMax = int(valores[i])

print "Nome: {0} - Nota Minima: {1} - Nota Maxima: {2}".\
format(valores[0], notaMin, notaMax)
linha = arq.readline()

arq.close()
```

4. Escreva um programa em Python que leia um arquivo de texto contendo uma lista de endereços IP e gere um outro arquivo, contendo um relatório dos endereços IP válidos e inválidos.
   O arquivo de entrada possui o seguinte formato:

```
200.135.80.9
192.168.1.1
8.35.67.74
257.32.4.5
85.345.1.2
1.2.3.4
9.8.234.5
192.168.0.256

arq = open('ips.txt', 'r')
validos = []
invalidos = []

linha = arq.readline()
while linha:

valores = linha.split('.')
valido = True

for i in range(0, len(valores)):
if(int(valores[i]) > 255):
valido = False
break

if(valido):
validos.append(linha)
else:
invalidos.append(linha)

linha = arq.readline()

arq.close()
arq = open('ips_saida.txt', 'w')

arq.write('[Enderecos validos:]\n')
for i in range(0, len(validos)):
arq.write(validos[i])

arq.write('\n[Enderecos invalidos:]\n')
for i in range(0, len(invalidos)):
arq.write(invalidos[i])

arq.close()
```

5. A ACME Inc., uma empresa de 500 funcionários, está tendo problemas de espaço em disco no seu servidor de arquivos. Para tentar resolver este problema, o Administrador

de Rede precisa saber qual o espaço ocupado pelos usuários, e identificar os usuários com maior espaço ocupado. Através de um programa, baixado da Internet, ele conseguiu gerar o seguinte arquivo, chamado "usuarios.txt":

```
alexandre       456123789
anderson        1245698456
antonio         123456456
carlos          91257581
cesar           987458
rosemary        789456125
```

Neste arquivo, o nome do usuário possui 15 caracteres. A partir deste arquivo, você deve criar um programa em Python que gere um relatório, chamado "relatorio.txt", no seguinte formato:

```
ACME Inc.               Uso do espaço em disco pelos usuários
------------------------------------------------------------------------
Nr.  Usuário        Espaço utilizado     % do uso

1    alexandre       434,99 MB            16,85%
2    anderson       1187,99 MB            46,02%
3    antonio         117,73 MB             4,56%
4    carlos           87,03 MB             3,37%
5    cesar             0,94 MB             0,04%
6    rosemary        752,88 MB            29,16%

Espaço total ocupado: 2581,57 MB
Espaço médio ocupado: 430,26 MB
```

O arquivo de entrada deve ser lido uma única vez, e os dados armazenados em memória, caso sejam necessários, de forma a agilizar a execução do programa. A conversão da espaço ocupado em disco, de bytes para megabytes deverá ser feita através de uma função separada, que será chamada pelo programa principal. O cálculo do percentual de uso também deverá ser feito através de uma função, que será chamada pelo programa principal.

```
def Converte(tam):
tamConv = float(tam)/float(1024*1024)

return round(tamConv,2)

def CalcularPorcentagem(espacoList, porcentagem):
espacoTotal = sum(espacoList)

for i in range(0, len(espacoList)):
perc = (espacoList[i] * 100)/float(espacoTotal)
porcentagem.append(round(perc,2))
```

```
arq = open('usuarios.txt', 'r')
nomesList = []
espacoList = []
porcentagem = []

nome = arq.read(15)
espaco = arq.readline()
while espaco:
nomesList.append(nome)
espacoList.append(Converte(espaco))

nome = arq.read(15)
espaco = arq.readline()

arq.close()

CalcularPorcentagem(espacoList, porcentagem)
espacoTotal = round(sum(espacoList),2)
espacoMedio = round(espacoTotal/len(nomesList),2)

print 'ACME Inc.               Uso do espaco em disco pelos usuarios'
print '-' * 69
print 'Nr.  Usuario        Espaco utilizado     % do uso'

for i in range(0, len(nomesList)):
print '{0} {1}{2}{3}%'.\
format(str(i + 1).ljust(4),nomesList[i],
(str(espacoList[i]) + ' MB').ljust(22),
porcentagem[i])

print '\nEspaco total ocupado: {0} MB'.format(espacoTotal)
print 'Espaco medio ocupado: {0} MB'.format(espacoMedio)
```

16.12 Para Saber Mais

Procure sobre o argumento opcional buffering do comando open e sobre outros métodos de arquivos.

Capítulo 17

Interfaces Gráficas e EasyGui

17.1 Interfaces Gráficas

As Interfaces gráficas (também chamadas de Graphical User Interfaces (GUI)) são usadas em aplicações modernas que requerem uma interação constante com o usuário. Assim permitindo maior usabilidade e naturalidade do que interfaces textuais.

Utilizando-as uma aplicação apresenta uma ou mais janelas com elementos gráficos que servem para comandar ações, especificar parâmetros, desenhar e exibir gráficos, etc.

Existem diversas bibliotecas (*toolkits*) para construção de interfaces, tais como:

- Qt
- Gtk
- wxWindows
- Tk

Python possui camadas de portabilidade (*bindings*) para várias bibliotecas de construção de interfaces.
Ex:

- PyQt (Qt)
- PyGtk (Gtk)
- wxPython (wxWindows)
- Tkinter (Tk)

Nesse capítulo nos focaremos no módulo EasyGui, por conta de sua finalidade didática.

17.2 EasyGui

EasyGui é um módulo que contém vários diálogos pré-definidos usando Tkinter. A ideia é usar esses diálogos como funções, retornando as escolhas feitas neles pelos usuários.

17.2.1 Onde Baixar

O EasyGui pode ser baixado através desse link: http://easygui.sourceforge.net/

17.2.2 Como Usar

Importe a biblioteca usando:

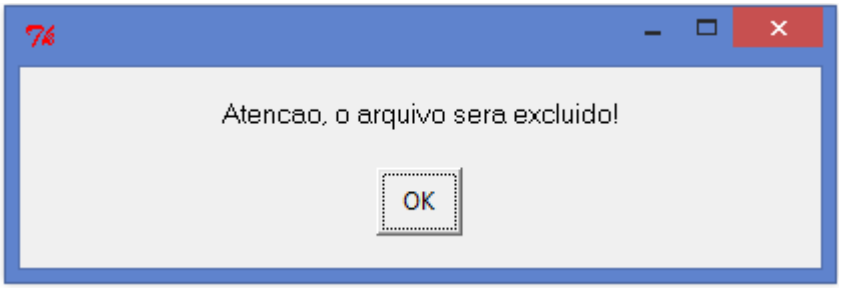
```
from easygui import * #importa toda biblioteca

import easygui as eg #importa caso-a-caso e chama de eg
```

17.2.3 Algumas Funções do EasyGui

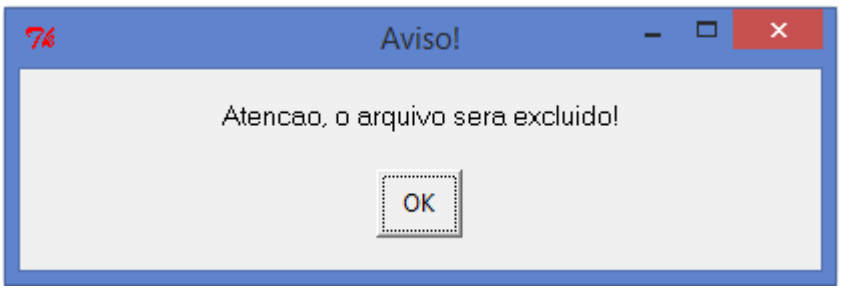
17.2.3.1 Caixas de Botões - Pré Definidas

msgbox
A caixa de mensagem msgbox exibe uma mensagem e oferece um botão OK. Pode-se enviar qualquer mensagem, junto com o título desejado. Pode-se sobreescrever o texto padrão de "OK"no botão se desejar.
Sintaxe: msg("sua mensagem",title='titulo',ok_button='ok')

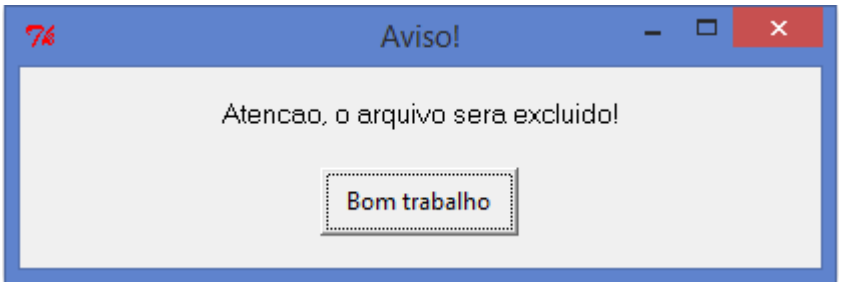
```
import easygui as eg
eg.msgbox("Atencao, o arquivo sera excluido!")
```




```
eg.msgbox("Atencao, o arquivo sera excluido!", "Aviso!")
```

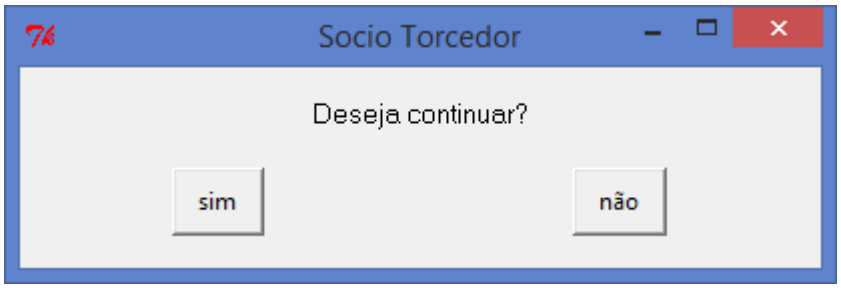



```
eg.msgbox("Atencao, o arquivo sera excluido!", "Aviso!",ok_button="Bom trabalho")
```



ccbox
Oferece a escolha para continuar ou cancelar
Sintaxe: ccbox(msg=Continua?', title=' seu titulo', choices=('Continua', 'Cancela'))

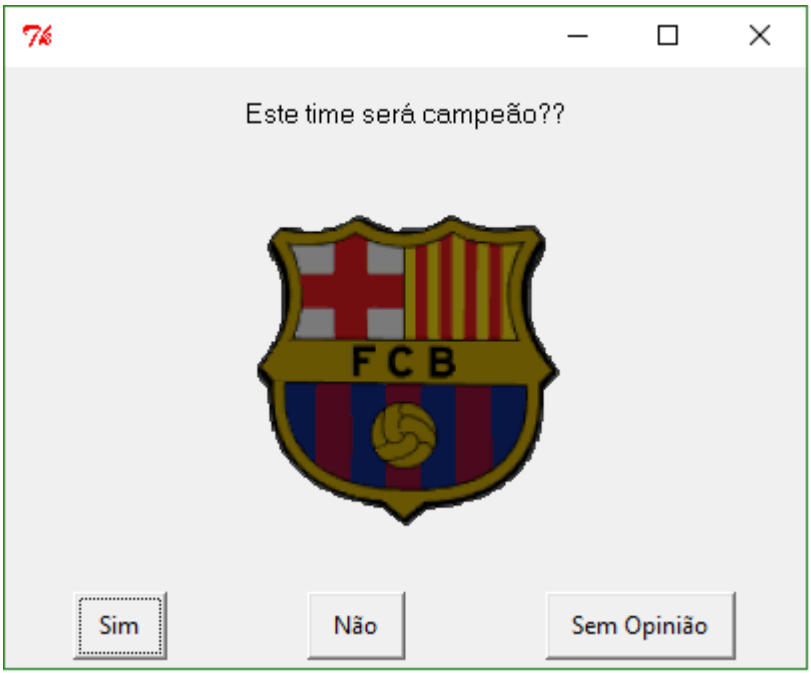
```
import easygui as eg
eg.ccbox("Deseja continuar?","Socio Torcedor",choices=('sim','não'))
```



17.2.3.2 Caixas de Botões Definidas Pelo Usuário

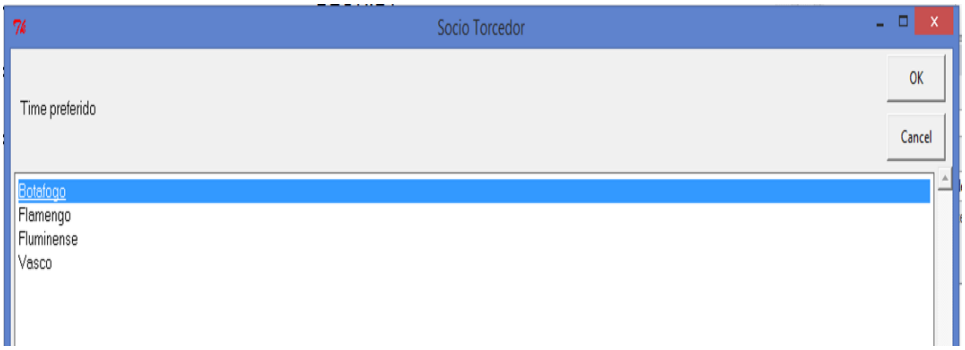
buttonbox
Buttonbox mostra uma mensagem, um titulo e um conjunto de botões. Retorna o texto do botão selecionado.
Sintaxe: variavel=buttonbox(msg='', title=' ', choices=('Button1', 'Button2', 'Button3'), image=None)

```
import easygui as eg
image = "f:/easygui/Barcelona.gif"
msg = "Este time será campeão??"
choices = ["Sim","Não","Sem Opinião"]
resp = eg.buttonbox(msg, image=image, choices=choices)
```



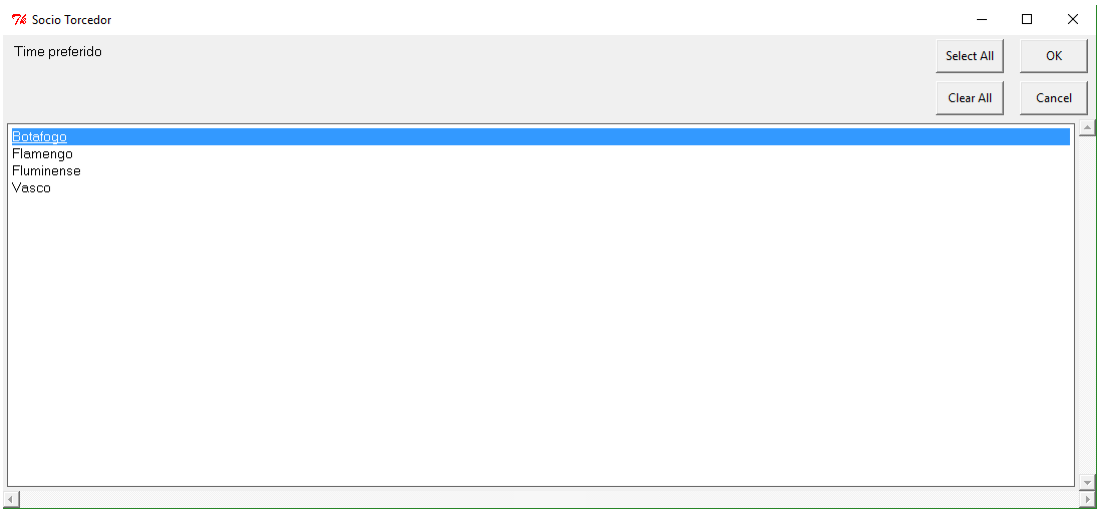
choicebox
Permite UMA escolha dentre múltiplas alternativas.
Sintaxe: choicebox(msg='Escolha algum.', title=' ', choices=(), buttons=())

```
import easygui as eg
escolha=['Flamengo','Fluminense','Vasco','Botafogo']
time=eg.choicebox('Time preferido',title='Socio Torcedor',choices=escolha)
```



multchoicebox
Entrada múltipla de dados. Retorna uma lista com as escolhas. Se o usuário não escolher nenhuma retorna uma lista vazia. Se o usuário escolher cancel retorna None.
Sintaxe: multchoicebox(msg=Escolha os itens', title=' ', choices=(), **kwargs)

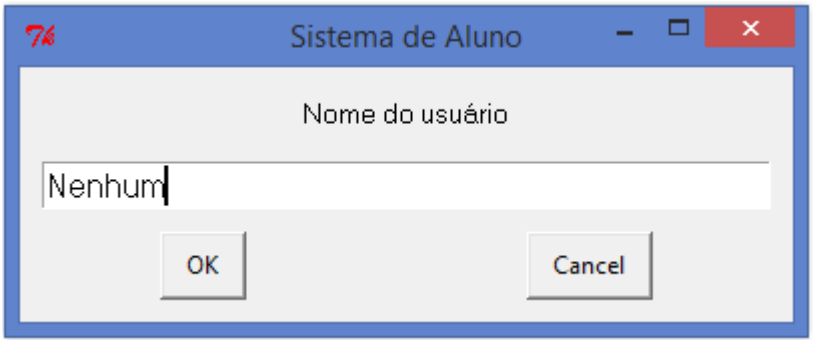
```
import easygui as eg
escolha=['Flamengo','Fluminense','Vasco','Botafogo']
time=eg.multchoicebox('Time preferido',title='Socio Torcedor',choices=escolha)
print time
```



17.2.3.3 Entrando com Informações

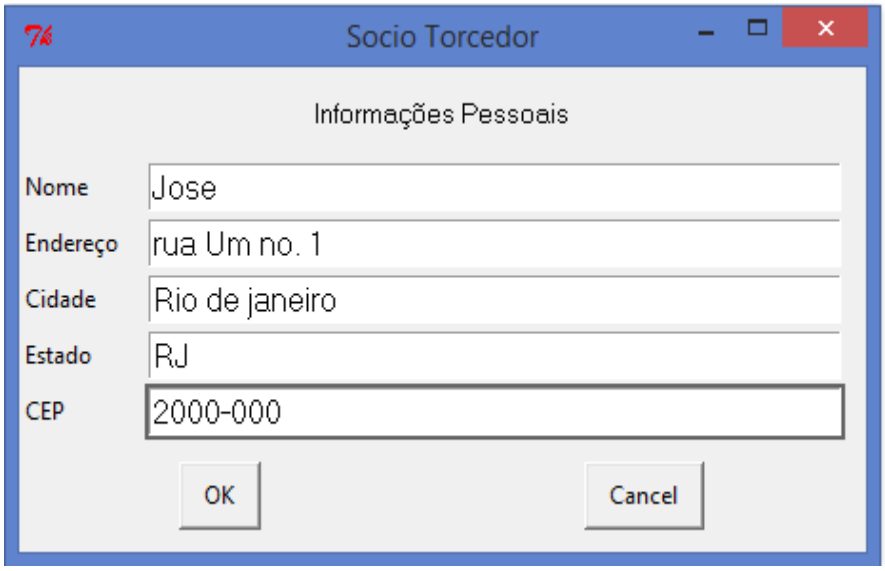
enterbox
Atribui um texto a uma variável. Mostra uma caixa para uma entrada simples de dados. Pode-se atribuir uma entrada padrão (default). O parâmetro strip remove os espaços em branco no inicio e no fim da entrada. Caso deseje-se manter coloca-se strip=False
Sintaxe: enterbox(msg='mensagem',title='titulo',default= 'padrão',strip=True)

```
import easygui as eg
nome=eg.enterbox('Nome do usuário','Sistema de Aluno',default='Nenhum',strip=True)
```



multenterbox
Retorna uma lista com os valores digitados ou None se selecionado cancel; Caso não seja digitado um campo retorna uma string vazia
Sintaxe: multenterbox(msg=preencha os campos', title=' ', fields=(), values=())

```
import easygui as eg
msg = "Informações Pessoais"
title = "Socio Torcedor"
campos = ["Nome","Endereço","Cidade","Estado","CEP"]
valores = eg.multenterbox(msg,title, campos)
print valores
```

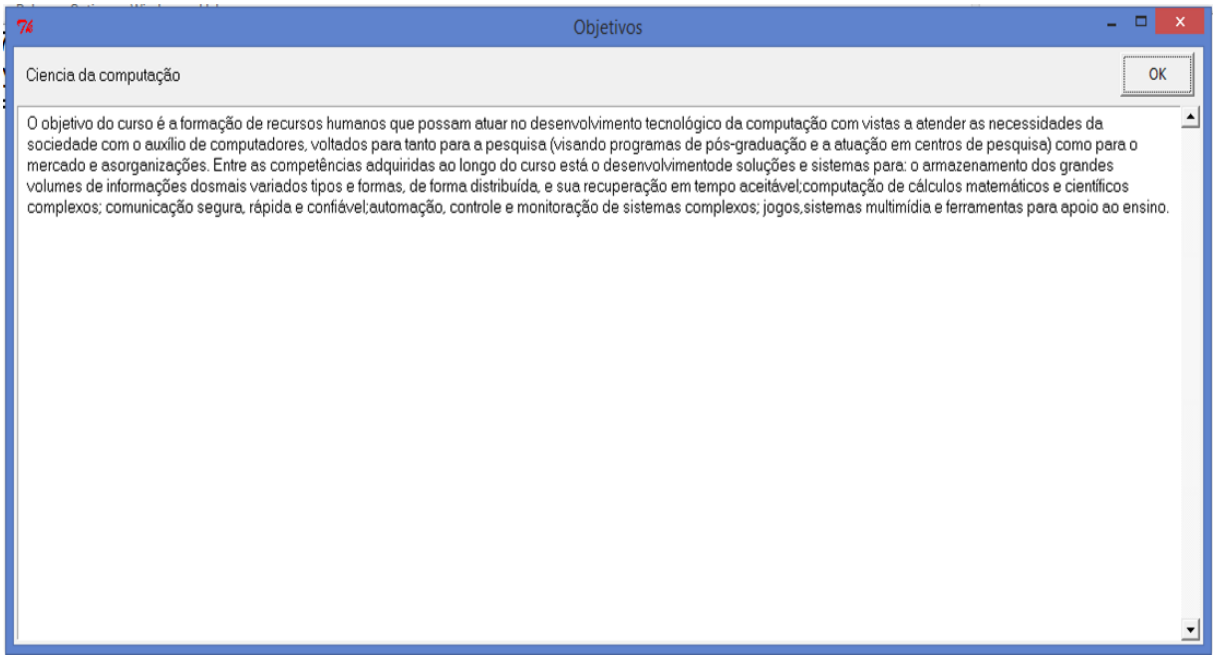



```
['Jose', 'rua Um no. 1', 'Rio de janeiro', 'RJ', '2000-000']
>>>
```

17.2.3.4 Mostrando um Texto

textbox
Mostra um texto com espaçamento proporcional e quebrando linhas. Codebox – indica se a fonte é proporcional ou não
Sintaxe: textbox(msg=”, title=’ ’, text=”, codebox=0)

```
import easygui as eg
msg="O objetivo do curso é a formação de recursos humanos \
que possam atuar no desenvolvimento tecnológico da computação com vistas
a atender as \
necessidades da sociedade com o auxílio de computadores, voltados para
tanto para a pesquisa \
(visando programas de pós-graduação e a atuação em centros de
pesquisa) como para o mercado e as\
organizações. Entre as competências adquiridas ao longo do curso está
o desenvolvimento\
de soluções e sistemas para: o armazenamento dos grandes volumes de
informações dos\
mais variados tipos e formas, de forma distribuída, e sua
recuperação em tempo aceitável;\
computação de cálculos matemáticos e científicos complexos;
comunicação segura, rápida e confiável;\
automação, controle e monitoração de sistemas complexos; jogos,\
sistemas multimídia e ferramentas para apoio ao ensino."
eg.textbox('Ciencia da computação','Objetivos',msg)
```



17.2.4 Exemplo de Aplicação

17.2.4.1 Objetivo

Manter o cadastro de socios-torcedores de um time de futebol.

17.2.4.2 Atores

A aplicação terá um único ator (pouco usual), o Gerente da Aplicação que cria, altera, remove sócios e emite relatórios.

17.2.4.3 Casos de Uso

- Criar o Banco de Dados BD e incluir sócios
- Incluir novos sócios
- Alterar dados do sócio
- Remover sócio do BD
- Emitir relatório com os sócios

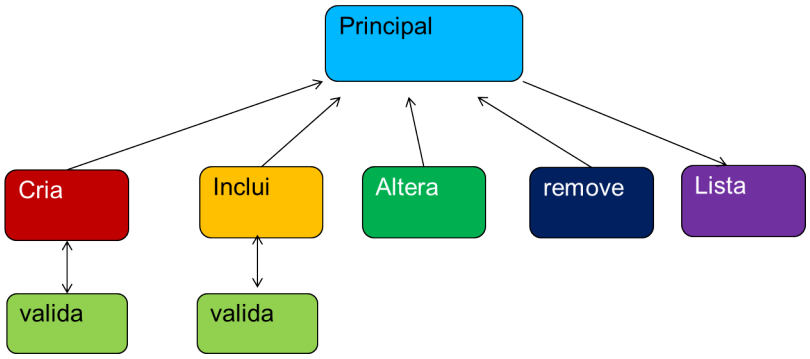
17.2.4.4 Estrutura de Dados

- Campos:
  - Nome
  - Endereço
  - Cidade

  - Estado
  - Cep

- Um único arquivo do tipo CSV (comma separeted values)
- A chave de busca será o nome do sócio, podendo existir um homônimo

17.2.4.5 Diagrama



17.2.4.6 Programa Principal

- Apresenta tela com as opções
- Abre o BD conforme a opção
- Executa a rotina selecionada
- Retorna a tela principal até o usuário optar por sair ou cancelar

```
def valida(campos,valores):
#validação dos campos
        if valores==None: #Se o usuario apertou cancela, sai do programa
                sys.exit(0)
        else:
                while True:
                errmsg = ""
                for i in range(len(campos)):
                        if valores[i] == "": # se o campo estiver vazio
                                errmsg += ('"%s" é um campo obrigatorio\
                                .\n\n' % campos[i])

                if errmsg == "": # sem erros
                        break
                else: # com erros
                        valores = eg.multenterbox(errmsg, title, campos,
                        valores)
```

```
        return

def legrava(arq):
        import sys

        msg = "Informações Pessoais"
        title = "Socio Torcedor"
        campos = ["Nome","Endereço","Cidade","Estado","CEP"]
        valores = []

        while True:
                valores = eg.multenterbox(msg,title, campos)
                valida(campos,valores)
                arq.write(valores[0]+','+valores[1]+','+valores[2]+','
                +valores[3]+','+valores[4]+'\n')
                msg1 = "Nome incluido \nVoce gostaria de continuar a\
                entrar dados de torcedores?"
                title1 = "Confirmacao"

                if eg.ccbox(msg1, title1,("sim","nao")):
                        pass # se a resposta for sim
                else:
                        break

        return

def lista(arq):
        dados=arq.readlines()
        dados.sort() #ordena por nome - primeiro campo

        print '\n'+'Arquivo'+'\n'
        print'{:' '10s}{:' '20s}{:' '10s} {:' '10s}{:' '15s}\n'.format\
        ('Nome','endereco','cidade','estado','   cep')

        for linha in dados:
                valores=linha.split(',')
                print '{:' '<10s}'.format(valores[0].ljust(10)),
                print '{:' '<20s}'.format(valores[1].ljust(20)),
                print '{:' '<10s}'.format(valores[2].ljust(10)),
                print '{:' '<10s}'.format(valores[3].ljust(10)),
                print '{:' '<10s}'.format(valores[4].ljust(10))

        a=raw_input('Pressione enter para encerrar-->')
        return

def remove(arq):
```

```
        import os

        auxi=open('baux.txt','w+')
        achei=False
        # procura sequencial em arquivo
        nomep=eg.enterbox(msg='Nome a remover', title=' ', default='',
        strip=True)

        for i in arq:
                dados=i.split(',')
                if dados[0]==nomep:
                        achei=True
                        eg.textbox(msg='Nome a ser removido', title=' ',
                        text=dados[0])
                else:
                        auxi.write(i) # grava se for diferente

        if achei:
                eg.msgbox(msg='(Nome removido)', title=' ', ok_button='OK')
        else:
                eg.msgbox(msg='(Nome não encontrado)', title=' ',
                ok_button='OK')

        arq.close() # fecha arq
        auxi.close() # fecha aux
        os.remove('arq1.txt') #apaga prog8.txt
        os.rename('baux.txt','arq1.txt')#renomeia arquivo
        return

def altera(arq):
        import os

        auxi=open('baux.txt','w+')
        achei=False
        # procura sequencial em arquivo
        nomep=eg.enterbox(msg='Nome a alterar', title=' ', default='',
        strip=True)

        for i in arq:
                dados=i.split(',')

                if dados[0]!=nomep:
                        auxi.write(i) # grava se for diferente
                else:
                        achei=True
                        msg = "Alteração das Informações Pessoais de "
```

```
                        +nomep
                        title = "Socio Torcedor"
                        campos = ["Endereço","Cidade","Estado","CEP"]
                        valores = eg.multenterbox(msg,title, campos)
                        valida(campos,valores)
                        #o nome não é alterado
                        #sao apenas 4 campos valores
                        auxi.write(nomep+','+valores[0]+','+valores[1]+','
                        +valores[2]+','+valores[3]+'\n')

        if achei:
                eg.msgbox(msg='(Nome alterado)', title=' ', ok_button='OK')
        else:
                eg.msgbox(msg='(Nome não encontrado)', title=' ',
                ok_button='OK')

        arq.close() # fecha arq
        auxi.close() # fecha aux
        os.remove('arq1.txt') #apaga arq1.txt
        os.rename('baux.txt','arq1.txt')#renomeia arquivo
        return

#programa principal
import easygui as eg
import sys

while True:
        msg ="Escolha a opcao desejada?"
        titulo = "Socio Torcedor"
        opcoes = ['Criar arquivo','Incluir Socio','Remover','Alterar',
        'Listar','Sair']
        escolha = eg.choicebox(msg, titulo, opcoes)

        if escolha=='Criar arquivo':
                arq=open("arq1.txt","w")
                legrava(arq)
                arq.close()
        elif  escolha=='Incluir Socio':
                arq=open("arq1.txt","a")
                legrava(arq)
                arq.close()
        elif escolha=='Listar':
                arq=open("arq1.txt","r")
                lista(arq)
                arq.close()
        elif escolha=='Remover':
```

```
        arq=open("arq1.txt","r")
        remove(arq)
        arq.close()
elif escolha=='Alterar':
        arq=open("arq1.txt","r")
        altera(arq)
        arq.close()
elif escolha =='Sair' or escolha==None: #Sai do Programa
        sys.exit(0)
```