TÉCNICAS DE PROGRAMAÇÃO

Engenharia da Computação – 2019.1 – UFC Sobral
Prof. Wendley S. Silva

Aulas 1 a 4

Observações importantes

Comunicação

wendley@ufc.br

www.ec.ufc.br/professores/wendley

Presença

- Há reprovações por falta!
- Altamente recomendado não faltar.
- Disciplina possui ritmo acelerado.

Início das aulas

- Segunda-feira: 08h00min
- Terça-feira: 08h00min

Evitemos atrasos!

Segundo o MEC, a duração da hora-aula nos cursos de graduação ministrados por estabelecimentos de ensino superior é 50 minutos.
http://www.guiadoestudante.ufc.br/base-de-informacoes/sistema-de-creditos

Recomendações

- Carga horária: 64 horas
- Estudem!
- Disciplina não é fácil.

- Use uma agenda (se possível).
- Fique menos tempo online!
- Pense antes de agir; aprenda a dizer “NÃO”.
- Cumpra o que prometeu

Monitores

2019.1:

- A definir (avisarei pelo Sigaa)

WELCOME TO ADVENTURE

Agenda

- Unidade 1 – Introdução ao Conceito de Subprogramas: modularização, passagem de parâmetros, variáveis locais e globais, recursividade.
- Unidade 2 – Aprofundamento nos Conceitos de Estruturas Básicas de Dados: vetores, matrizes e registros.
- Unidade 3 – Variáveis Dinâmicas: conceito de ponteiros, alocação dinâmica de memória e aplicação associada às estruturas básicas de dados (listas, pilhas e filas).
- Unidade 4 – Abstração: encapsulamento e proteção, interface e implementação.

Agenda

- Unidade 5 – Programação Estruturada: modularização, estruturas de seleção, decisão e repetição.
- Unidade 6 – Refinamentos Sucessivos: estratégias e padrões de desenvolvimento.
- Unidade 7 – Manipulação de Arquivos: conceito de arquivos (sequencial, randômico), funções de manipulação de arquivos em Java.
- Unidade 8 – Programação Orientada a Objetos: conceitos e implementação de classes, objetos, variável de instância, método, herança, construtores, destrutores, sobrecarga, polimorfismo e exceções.

Agenda

- Linguagens

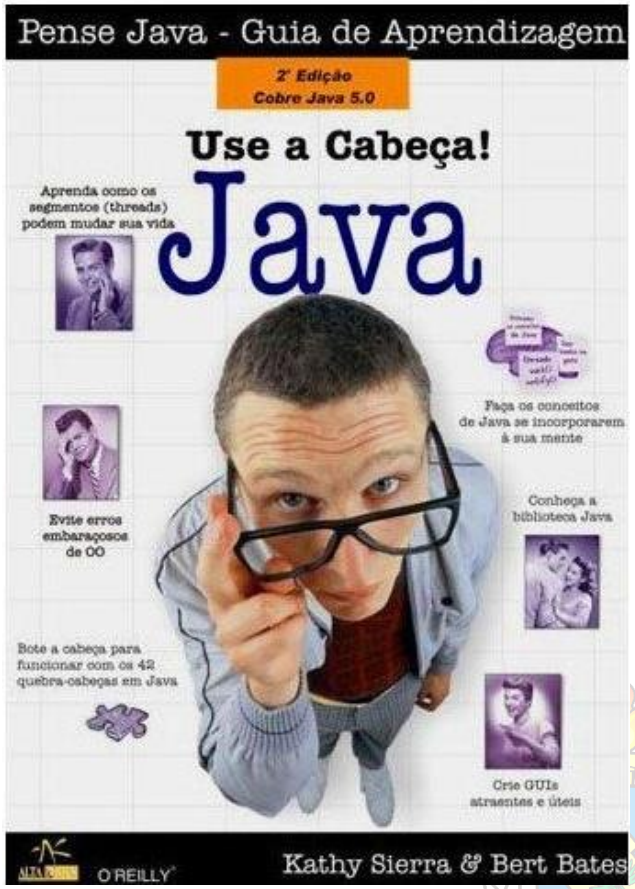
Agenda

- A linguagem Java
  - *Introdução*
  - *Codificação*
  - *Fundamentos*

Livros

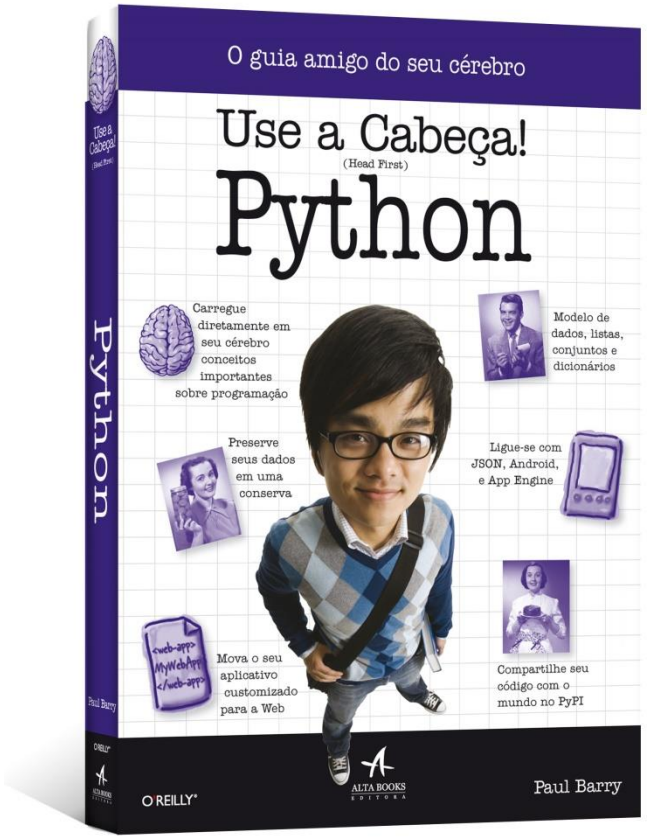

NOVA EDIÇÃO ATUALIZADA
JAVA™ COMO PROGRAMAR
6ª Edição
Destacando INTRODUÇÃO A CLASSES E OBJETOS
OOD COM A UML™
J2SE™ 5.0
DEITEL
DEITEL
PEARSON
Prentice Hall
DEITEL®


Java™
COMO PROGRAMAR
8ª edição
DEITEL®
PAUL DEITEL
HARVEY DEITEL
PEARSON


Pense Java - Guia de Aprendizagem
2ª Edição
Cobre Java 5.0
Use a Cabeça!
Java
Aprenda como os segmentos (threads) podem mudar sua vida
Faça os conceitos de Java se incorporarem à sua mente
Evite erros embaraçosos de OO
Conheça a biblioteca Java
Bote a cabeça para funcionar com os 42 quebra-cabeças em Java
Crie GUIs atraentes e úteis
O'REILLY
Kathy Sierra & Bert Bates

Livros


2ª EDIÇÃO
Revisada e ampliada
INTRODUÇÃO À PROGRAMAÇÃO COM
PYTHON
Algoritmos
e lógica de programação
para iniciantes
novatec
Nilo Ney Coutinho Menezes


O guia amigo do seu cérebro
Use a Cabeça!
(Head First)
Python
Carregue diretamente em seu cérebro conceitos importantes sobre programação
Modelo de dados, listas, conjuntos e dicionários
Preserve seus dados em uma conserva
Ligue-se com JSON, Android, e App Engine
Mova o seu aplicativo customizado para a Web
Compartilhe seu código com o mundo no PyPI
O REILLY
ALTA BOOKS
Paul Barry

Sistema de notas

- Média = MediasAPs / QtdeAPs
- MediaAP1 = 0,5 x AP1 + 0,35 x Trabalhos práticos + 0,15 x Exercícios em Lab.

- Uma prova teórica e três provas práticas *

** Expectativa*

Por que programar?

- Quais ações do dia a dia são ou podem ser controladas por computadores?
- O que você gostaria de ensinar o computador a fazer?,
- O que você acha que é preciso para programar um robô?


Java™ starting...

Tecnologia Java

- Lançada em 1995, hoje é uma plataforma muito estável e madura
- É baseada na abrangência da internet e na ideia de que um *software* deve ser capaz de rodar em diferentes máquinas, sistemas e dispositivos.
- Os programas feitos em Java rodam em diferentes ambientes graças a um componente da plataforma chamado JVM (*Java Virtual Machine*)

Tecnologia Java

- A Tecnologia Java é basicamente dividida em:
  - *JSE – Java Standard Edition*
  - *JEE – Java Enterprise Edition*
  - *JME – Java Micro Edition*

- Java é:
  - *uma completa plataforma de soluções para tecnologia*
  - *um ambiente de desenvolvimento*
  - *uma linguagem de programação*

JSE - Java Standard Edition

- JSE permite o desenvolvimento de aplicações Java para desktop ou cliente/servidor

- É composta por:
  - ***JRE:*** *provê a API Java, a Java Virtual Machine e outros componentes necessários para rodar aplicações escritas na Linguagem Java.*
  - ***JDK:*** *contém tudo o que há na JRE além de ferramentas como compiladores e debugadores necessários para desenvolver as aplicações*

JEE e JME

- **JEE**: é o padrão para desenvolvimento de sistemas corporativos e é voltada para aplicações multi-camadas, baseadas em componentes executados em servidores de aplicações

- **JME**: é o padrão aplicado a dispositivos compactos, como celulares, PDAs, controles remotos, e uma gama de dispositivos. Permite o desenvolvimento de software embarcados

IDEs

- IDEs para o desenvolvimento em Java:
  - *Eclipse*
    - www.eclipse.org
  - *NetBeans*
    - www.netbeans.org

Introdução à Linguagem Java

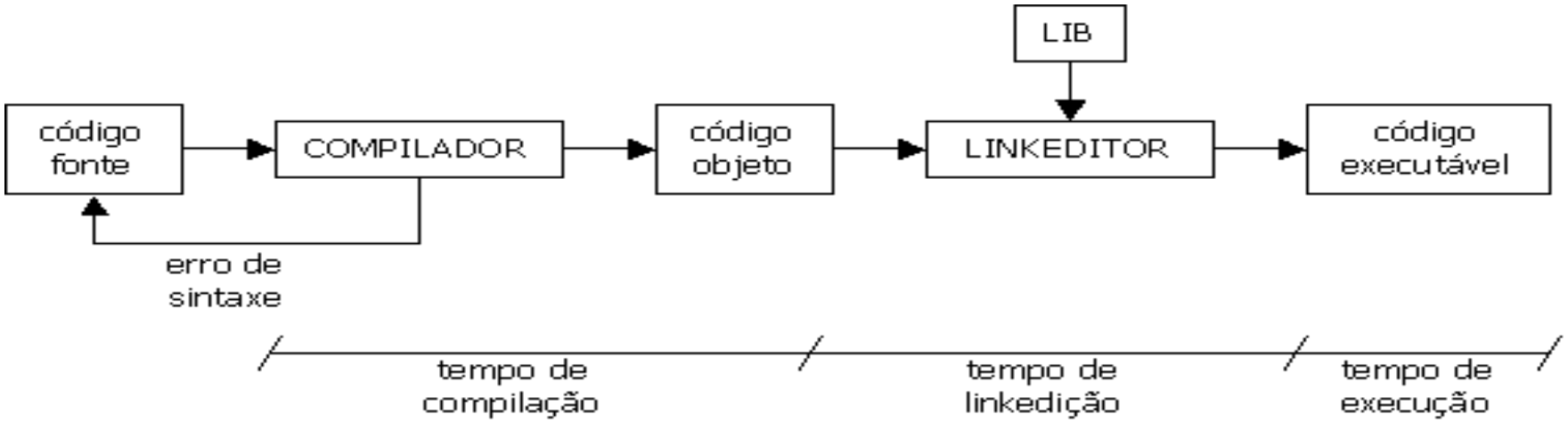
- A linguagem Java foi designada para ser:
  - ***Orientada a Objetos:*** *permite a criação de classes, herança e polimorfismo. A interação do sistema se dá pela interação dos objetos nele criados*
  - ***Distribuída:*** *dá suporte a aplicações distribuídas (RMI, CORBA, URL)*
  - ***Simples:*** *não permite a manipulação direta da memória (ponteiros em C), e possui o garbage collector para remover objetos não referenciados*

Introdução à Linguagem Java

- A linguagem Java foi designada para ser:
  - ***Multithread**: permite a execução de diversas tarefas ao mesmo tempo*
  - ***Segura**: possui medidas de segurança para proteger o código de ataques (não usa ponteiros)*
  - ***Independente de plataforma**: Write Once, Run Everywhere*

Introdução à Linguagem Java

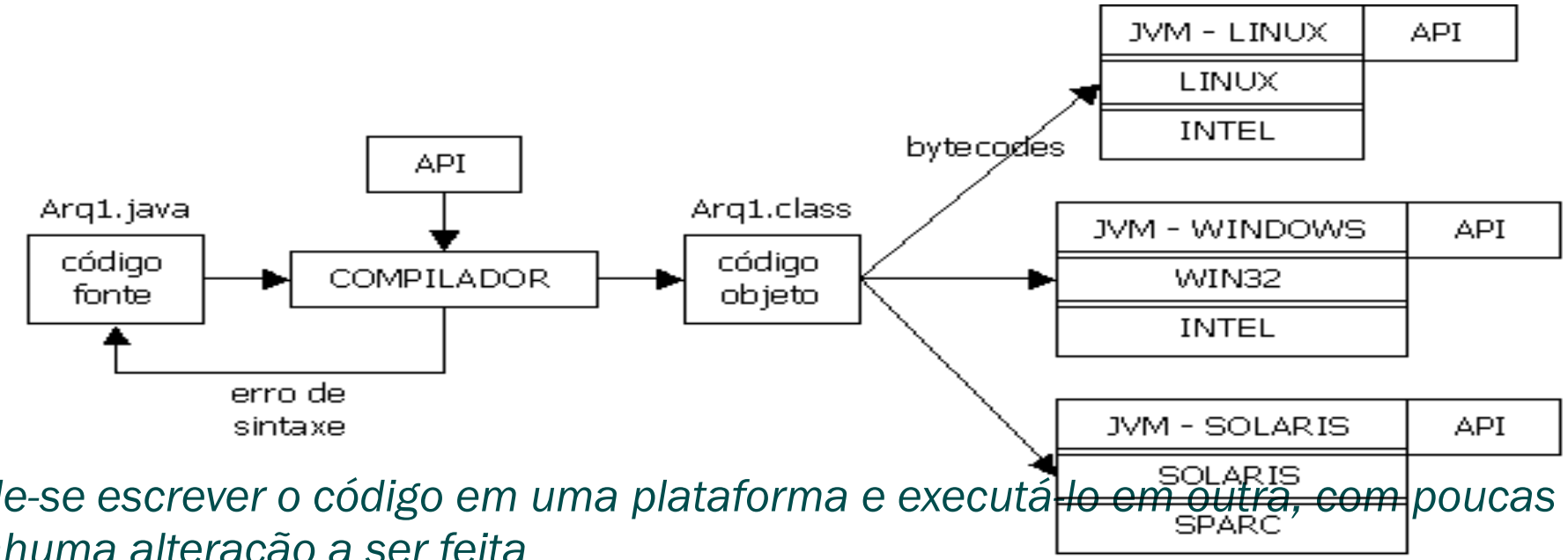
- Dependência de Plataforma
  - ***compilação*** *de código na linguagem C, C++*



  - *Para rodar em diferentes plataformas, o código em C deve ser compilado em diferentes compiladores*
  - *Código proprietário*

Introdução à Linguagem Java

- Independência de Plataforma
  - *JAVA: processos de **compilação** e **interpretação***



  - *Pode-se escrever o código em uma plataforma e executá-lo em outra, com poucas ou nenhuma alteração a ser feita*

Introdução à Linguagem Java

- Independência de Plataforma:
  - *Com o compilador, primeiramente o programa é convertido em **bytecodes Java** – um código independente de plataforma que é interpretado na plataforma Java.*
  - *O interpretador lê e executa cada instrução em bytecode no computador.*
  - *O código é compilado uma única vez; a interpretação ocorre a cada execução da aplicação e é feita pela JVM.*

Plataforma Java

- A plataforma Java difere das demais por ser apenas uma plataforma de software que roda sobre qualquer plataforma de hardware.

- Possui dois componentes:
  - ***Java Virtual Machine (JVM):*** *base da plataforma Java e portável para várias plataformas de hardware. Parte de software que executa código, tarefa normalmente efetuada pela CPU.*

  - ***Java Application Programming Interface (Java API):*** *grande coleção de componentes de software prontos que provêm diversas capacidades, como ambiente gráfico e funcionalidades específicas*

CODIFICAÇÃO

Detalhes iniciais

- Java é uma linguagem fortemente tipada.
- Não existem variáveis globais ou ponteiros.
- Java é case-sensitive:
  - *variavel ≠ Variavel ≠ VARIAVEL*
- Boas práticas:
  - *tabular corretamente o código*
  - *usar padrões de nomenclatura*
  - *usar nomenclatura clara e inteligível*
  - *comentar o código*

Detalhes iniciais

- Regras de nomenclatura:
  - ***não iniciar nomes com números ou caracteres especiais***
  - ***classes:*** *inicial maiúscula*
    - String, Vector, AcervoGeral, ControleContas
  - ***atributos, métodos e variáveis:*** *tudo em minúsculas; se houver mais de uma palavra, primeira palavra toda em minúsculas, demais palavras com inicial maiúscula*
    - nome, anoNascimento, getNome( ), setSalarioInicial( )
  - ***pacotes:*** *tudo em minúsculas*
    - java.util, java.lang, javax.swing
  - ***constantes:*** *tudo em maiúsculas*
    - PI, E
- Normalmente não se usam os caracteres "_" e "-" no nome de classes, variáveis ou métodos.

Detalhes iniciais

- Blocos de código são delimitados por “{” e “}” e definem o escopo das variáveis.
- O nome da classe deve ser idêntico ao nome do arquivo .java (considerando maiúsculas/minúsculas).

Arquivo: Livro.java

```
public class Livro {
   private String autor;
   private String titulo;

   public Livro(String autor,
setAutor(autor);
setTitulo(titulo);
   }
}
```

Primeiro programa

```
public class PrimeiroPrograma{
      public static void main(String[] args){
        // amostragem na tela
     System.out.println("Primeiro programa");
     }
}
```

define o início da classe

1º método a ser executado (deve ser mantida esta linha sem alterações)

- Este código deve ser escrito no arquivo **PrimeiroPrograma.java**.
- Ao ser compilado, irá gerar o arquivo de bytecodes **PrimeiroPrograma.class**.
- O arquivo .class é interpretado pela JVM e gera a execução do programa.
- Na tela, deve aparecer a mensagem "Primeiro Programa".

Análise do Primeiro Programa

```
public class PrimeiroPrograma
```

public: modificador de acesso (acesso público)
class: define o início da classe
PrimeiroPrograma: nome da classe

```
public static void main(String[] args)
```

(este método é o ponto de partida do programa)
public: modificador de acesso (acesso público)
static: método pertence à classe
void: método não possui retorno
main: nome do método
String[] args: parâmetro de entrada do método, array de Strings

```
System.out.println("Primeiro programa");
```

System.out.println: método de amostragem na tela
"Primeiro programa": mensagem a ser mostrada na tela

Análise do Primeiro Programa

- A classe que inicia a execução de um programa stand-alone deve possuir o **método main.**

- Comentários:

```
// comentário de uma linha

/*  bloco de
    comentários com
     mais de uma linha */

/** bloco de
 *  comentários
 *  formato javadoc **/
```

FUNDAMENTOS DA LINGUAGEM JAVA

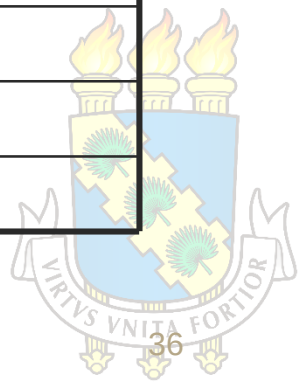
Criação de variáveis

- Variáveis são utilizadas nas linguagens em geral para armazenar um determinado valor.
- **Toda variável deve ser declarada e inicializada.**
  - *a declaração define o tipo da variável*
    - <tipo da variável> <nome da variável>;
  - *a inicialização define o seu valor inicial*
    - <nome da variável> = valor;
    - <tipo da variável> <nome da variável> = valor;
- Uma vez criada uma variável, de acordo com um tipo definido, este tipo não pode ser alterado.

Tipos Básicos

- Existem 8 tipos básicos (primitivos) em Java:

	Tipo	Tamanho (bits)	Mínimo	Máximo	Default	Sem sinal
Lógico	**boolean**	1/16	false	true	false	x
Caracter	**char**	16	0	$2^{16}-1$	0	x
Inteiro	short	8	-2^{7}	$2^{7}-1$	0	
	byte	8	-2^{15}	$2^{15}-1$	0	
	int	32	-2^{31}	$2^{31}-1$	0	
	long	64	-2^{63}	$2^{63}-1$	0	
Decimal	float	32	7 casas decimais		0.0	
	double	64	15 casas decimais		0.0	



String

- String não é um tipo básico, é uma classe do Java usada para representar textos (cadeias de caracteres).
- A classe String possui métodos que permitem a manipulação da sua cadeia de caracteres.

```
public class TesteStrings {
    public static void main(String[] args) {
        // declaração (s vale null)
        String s;
        // inicialização
        s = "Ana dos Santos";
        // declaração e inicialização
        String s2 = "Roberto Padilha";

        System.out.println("Igual: " + s.equals("Ana dos Santos"));    System.out.println("Tamanho de
    s: " + s.length());
        System.out.println("Caracter da posição 5 de s2: " + s.charAt(5));
        }
}
```

Operadores

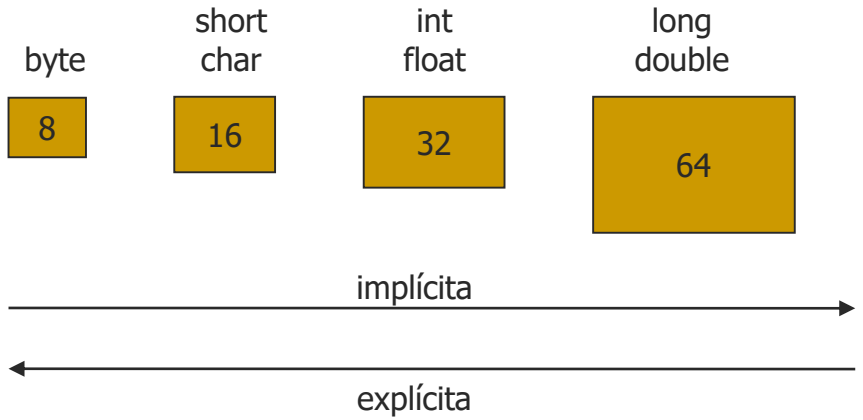
Aritméticos	Relacionais	Lógicos	Bit a bit
+ += ++	> >=	! (negação)	& (e)
- -= --	< <=	&& (e)	| (ou)
* *=	!=	|| (ou)	^ (xor)
/ /=	==		~ (negação lógica)
% %= (resto da divisão)			

```
int a = 5;
boolean b = (a == 10);
System.out.println(!b);
int c = 10 % 3;
System.out.println(c);
double d = 15.3;
d += 10;
System.out.println(d);
boolean e = (d > 10 && d <= 50)
System.out.println(e);
```

```
int i = 1;
int j = 2;
int k = ++j;
System.out.println("K: " + k);
System.out.println("J: " + j);
int l = i++;
System.out.println("L: " + l);
System.out.println("I: " + i);
k++;
```

Conversões

- As conversões podem ser
  - ***implícitas:*** *tipo menor para tipo maior*
  - ***explícitas:*** *tipo maior para tipo menor. De ponto flutuante para inteiro pode haver perda de informação.*



Conversões

- **Entre tipos básicos**

- Se há tipos diferentes em uma expressão, é feita a conversão implícita para o maior tipo da expressão.
  - `double a1 = 10 * 5.2 + 4 - 1.3f;` `a1 = 54.7`
  - `double a2 = 5 / 2;` `a2 = 2.0`
  - `double a3 = 5 / 2.0;` `a3 = 2.5`

- A conversão explícita é marcada no código (também chamada *casting*):
  - `int a4 = (int) 10 * 5.2 + 4 - 1.3f;` `a4 = 54`
  - `int a5 = (int) 5 / 2.0;` `a5 = 2`

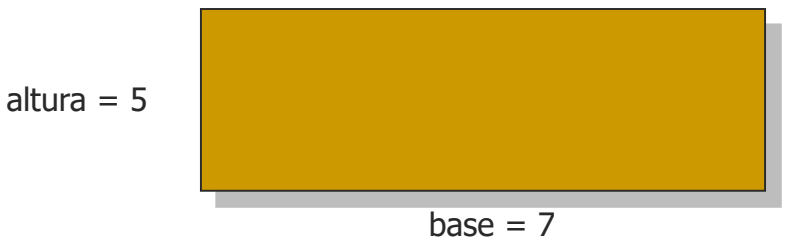
Conversões

- **Entre String e valor numérico:**
  - *como String não é um tipo básico, é necessário utilizar classes do Java que fazem a conversão de um texto em um número.*
  - ***Classes Wrapper:*** *classes que representam os tipos básicos:*
    - Character
    - Byte, Short, Integer, Long
    - Float, Double

```
String s = 10;
int a1 = Integer.parseInt(s);          a1 = 10
float a2 = Float.parseFloat(s);        a2 = 10.0f
double a3 = Double.parseDouble(s)          a3 = 10.0
```

Exemplo de Código

- Elaborar uma classe que contenha métodos para:
  - *calcular a área de um retângulo*
  - *calcular o perímetro de um retângulo.*
- Executar o código de modo que o área e o perímetro de um retângulo de base 7 e altura 5 sejam mostrados na tela.



- **Área**:
  - base x altura
- **Perímetro**:
  - 2 x base + 2 x altura

Exemplo de Código

```
public class TesteRetangulo {
     public static double calculaArea(double base, double altura){
     double area = base * altura;
     return area;
     }

     public static double calculaPerimetro(double base, double altura){
     return 2 * base + 2 * altura;
     }

     public static void main(String[] args){
     int b = 7;
     int h = 5;
     double a = calculaArea(b,h)

     System.out.println("Area: " + a);
     System.out.println("Perimetro: " + calculaPerimetro(b,h));
     }
}
```

Análise do Código

- A classe TesteRetangulo é composta por 3 métodos:
  - *calculaArea, calculaPerimetro, main*

- Assinatura de um método:

<modificadorDeAcesso> <static> tipoDeRetorno nomeDoMetodo (<listaDeParâmetros>)

```
public static double calculaArea(double base, double altura)
public static void main(String[] args)
```

  - ***modificadorDeAcesso:*** *delimita visibilidade do método (opcional)*
  - ***static:*** *não é necessário criar objeto para chamar o método, o método pertence à classe (opcional)*
  - ***tipoDeRetorno:*** *tipo básico, tipo abstrato (classe), ou void quando não há retorno*
  - ***listaDeParâmetros:*** *o que entra no método; cada parâmetro deve ter um tipo associado (opcional)*

Análise do Código

- Primeiramente é executado o método main onde são criadas as variáveis **b** e **h**, que recebem os valores 7 e 5 respectivamente.
- Ao chamar o método calculaArea(b, h), há um desvio para o método calculaArea, onde os parâmetros de entrada **base** e **altura** recebem os valores de **b** e **h**, respectivamente.
- Ao entrar no método calculaArea, é criada a variável **area** que assume o valor da expressão **base x altura** (35). O valor desta variável é retornado à variável **a** do método main.
- A variável **a** é apresentada na tela.
- Processo semelhante ocorre na chamada do método calculaPerimetro(b,h).

Análise do Código

- Escopo:
  - *as variáveis b, h e a são visíveis apenas no método main (variáveis locais do método main)*
  - *os parâmetros base e area e a variável area são visíveis somente no método calculaArea*
  - *os parâmetro base e area e a variável perímetro são visíveis somente no método calculaPerimetro.*
  - *ao término da execução de um métodos, seus parâmetros e variáveis locais são apagados da memória.*

- Por mais que haja dois métodos com parâmetros de nomes idênticos, os momentos de execução são diferentes. Por isso, não há confusão entre os parâmetros.


VIRTVS VNITA FORTIOR

```
#include <stdio.h>


void main(void){
    int t, i, num[3][4];
    char s[80];
    for(t=0; t<3; t++)
        for(i=0; i<4; i++){
            printf("Digite o valor em [%d,%d]:",t,i);
            gets(s);
            num[t][i]=atoi(s);
            printf("%3d",num[t][i]);
            printf("\n");
        }
}
```

```
package primeiraAula;
import java.io.*;
public class Teste {
    public static void main(String args[]){
     try{
                int t, i;
                int num[][] = new int[3][4];
                String s = "";
                for(t = 0; t < 3; t++)
                    for(i = 0; i < 4; i++){
                        System.out.print("Digite o valor em ["+t+","+i+"]:");
                        BufferedReader entrada = new BufferedReader( new InputStreamReader(System.in)
          );
                        s = entrada.readLine();
                        num[t][i] = Integer.parseInt(s);
                        System.out.println(num[t][i]);
                    }
              }catch(IOException e){}
   }
}
```

BufferedReader
IOException
InputStreamReader

```
package primeiraAula;

import java.util.Scanner;

public class Teste {

    public static void main(String args[]){
        int t, i;
       int num[][] = new int[3][4];
       Scanner input = new Scanner(System.in);
       for(t = 0; t < 3; t++)
          for(i = 0; i < 4; i++){
             System.out.print("Digite o valor em ["+t+","+i+"]:");
             num[t][i] = input.nextInt();
             System.out.println(num[t][i]);
          }
        }

}
```

Conhecimento básico de Java

- Todo programa funciona a partir de classes
- Todo programa tem um ponto de entrada
  - *Método **main** de uma classe pública*
- Tipos de dados:
- Numérico
  - *Inteiro: byte, short, int, long*
  - *Real: float, double*
- Literal: char
- Alfanumérico: String
- Lógico: boolean
- Constantes: public static final <tipo>

Conhecimento básico de Java

- Objeto: representação de algo presente no mundo real
- Pertence a uma classe:
  - *Objeto -> Fusca*
  - *Classe -> Automóvel*
- Possui:
  - *Dados -> cor, tamanho, peso*
  - *Comportamentos -> andar, parar*
- Inicialização de um objeto:
  - *método construtor*
  - *inicializa as variáveis objeto da classe*
  - *Carro fusca = new Carro();*

Estruturas de Seleção

- Seleção simples - IF:

```
if (condição)
    instrução quando condição é verdadeira
```

ou

```
if (condição)
    instrução quando condição é verdadeira
else
    instrução quando condição é falsa
```

Estruturas de Seleção

- Seleção simples - IF:

```
if (condição) {
    instruções para
    quando condição é verdadeira
} else {
    instruções para
    quando condição é falsa
}
```

Estruturas de Seleção

- Seleções simples aninhadas- IF:

```
if (condição1)
    instrução quando condição é verdadeira
else { // quando condição1 é falsa
    if (condição2)
        instrução quando condição2 é verdadeira
    else
        instrução quando condição2 é falsa
}
```

Estruturas de Seleção

- Seleção Múltipla - Switch:

```
int opcao = 2;
switch(opcao){
        case 1: mod1(); break;
        case 2: mod2(); break;
        case 3: mod3(); break;
        default: System.out.println(“Erro”);
}
```

Estruturas de Repetição

- WHILE – Teste no Início:

```
int quadrado = 2;
while ( quadrado <= 1000 )
      quadrado = 2* quadrado;
```

- DO-WHILE – Teste no Final:

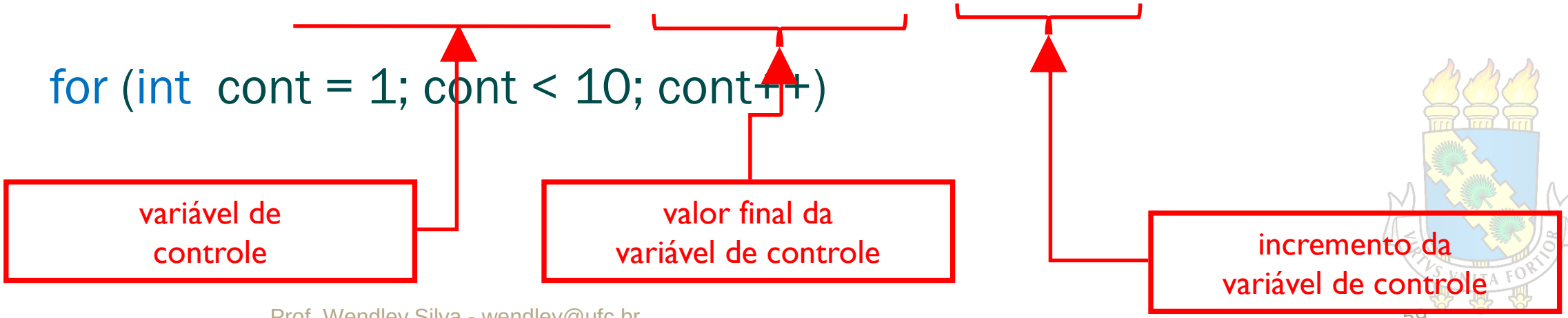
```
int contador = 1;
do{
    System.out.println( contador );
    contador++;
}while (contador <20)
```

Estruturas de Repetição

- FOR – Controle por Contador:

```
for (expressão1; expressão2; expressão3)
    instrução;
```

```
for (int  cont = 1; cont < 10; cont++)
```



Operadores de Atribuição

- Suponha: int c = 3, d = 5, e = 4, f = 6, g = 12

c += 7 => c = c + 7 => 10
d -= 4 => d = d - 4 => 1
*e *= 5 => e = e * 5 => 20*
f /= 3 => f = f / 3 => 2
g %= 9 => g = g % 9 => 3

?
?
?
?
?

Operadores de Incremento e Decremento

pré-incremento	++a	Incrementa **a** por **1**, depois utiliza o novo valor de **a** na expressão em que **a** reside.
pós-incremento	a++	Utiliza o valor atual de **a** na expressão em que **a** reside, depois incrementa **a** por **1**.
pré-decremento	--b	Decrementa **b** por **1**, depois utiliza o novo valor de **b** na expressão em que **b** reside.
pós-decremento	b--	Utiliza o valor atual de **b** na expressão em que **b** reside, depois incrementa **b** por **1**.

```
int c = 5;
System.out.println(c++);   → 5
System.out.println(++c);   → 6
```

Jogo dos 7 (sete) erros

```
public class Calcular{
    public class void main(String args[]){
        int soma = 0, x;
        x = 1.0;
        while (x <
            soma
            ++x;
        }
        Systema.out.println("O so atório é: " soma);
    }
}
```

Algoritmo que calcula e imprime a soma dos inteiros de 1 a 10 utili-
zando-se da espressão ***while*** para
os cálculos.

Jogo dos 7 (sete) erros

```
public class Calcular{
    public class void main(String args[]){
        int soma = 0, x;
        x = 1.0;
        while (x < 10) {
            soma += x;
            ++x;
        }
        Systema.out.println("O somatório é: " soma);
    }
}
```

Jogo dos 7 (sete) erros

```
public class Calcular{
    public static void main(String args[]){
        int soma, x;
        x = 1; soma = 0;
        while (x <= 10) {
            soma += x;
            ++x;
        }
        System.out.println("O somatório é: " + soma);
    }
}
```

Arrays

- Declarando e alocando arrays

```
int c[] = new int[12];
```

```
int c[];
c = new int[12];
```

```
int vetor[] = {10, 20, 30, 40, 50};
```

Arrays

- Passando *arrays* para métodos

```
int  temperatura[]  =  new  int[25];

void  modificaArray ( int  b[] ) {  (...)  }

modificaArray ( temperatura );
```

Arrays Multidimensionais

- São mantidos como *arrays* de *arrays*

- *Array* bidimensional: b[2][2]

```
int  b[][]  =  {  { 1, 2 },  { 3, 4 }  };
System.out.println("Valor: "+b[1][1]);
int  b[][]  =  { { 1, 2 }, { 3, 4, 5 } };  (?)
System.out.println("Valor: "+b[0][2]);
```

Arrays Multidimensionais

```
int  b[][];
b  =  new  int[3][4];

int b[][];
b  =  new  int[2][];  //  aloca linhas
b[0]  =  new  int[5];  // aloca colunas para a linha 0
b[1]  =  new  int[3];  // aloca colunas para a linha 1
```

Aplicação -> Histograma

- Ferramenta gráfica para verificar a freqüência de valores de um conjunto

- Histograma para exibir dados de um *array*
  - *[19, 3, 15, 7, 11, 9, 13, 5, 17, 1]*

```
public class Histograma {

    public static void main(String args[]){
        int array[] = {19, 3, 15, 7, 11, 9, 13, 5, 17, 1};

        String saida = "Elemento\tValor\tHistograma";

        // para cada elemento do array, desenha uma barra no histograma
        for(int cont = 0; cont < array.length; cont++){
            saida += "\n"+cont+"\t"+array[cont]+"\t";
            for(int estrela = 0; estrela < array[cont]; estrela++) saida+= "*";
        }

        System.out.println(saida);
    }
}
```

Histograma

- Saída:

Elemento	Valor	Histograma
0	19	**********
1	3	***
2	15	**********
3	7	*******
4	11	**********
5	9	*********
6	13	**********
7	5	*****
8	17	**********
9	1	*

Operadores Lógicos

Operadores	Significado
== !=	igualdade/desigualdade
&	E lógico booleano
^	OU lógico booleano exclusivo (XOR)
|	OU lógico booleano
&&	E lógico
||	OU lógico

George Boole (02/11/1815 – 08/12/1864).
Matemático e filósofo britânico. É o criador da Álgebra Booleana, base da atual aritmética computacional.

Tabela-verdade

A	B	!A	A && B	A || B	A ^ B
1	1	F	1	1	F
1	0	F	0	1	V
0	1	V	0	1	V
0	0	V	0	0	F

função de **mínimo**

função de **máximo**

(!A && B) + (A && !B) ?

Conversão de tipos de dados

String num = “5”; String -> int, float, double

```
int x = Integer.parseInt(num);
float y = Float.parseFloat (num);
double z = Double.parseDouble(num);
```

int, float, double -> String

```
num = String.valueOf(x);
num = String.valueOf(y);
num = String.valueOf(z);
```

Revisão

- Classes e Objetos
- Linguagem Java
  - *Tipos de Dados*
  - *Estruturas de Seleção e Repetição*
  - *Operadores*
  - *Arrays*
  - *Conversão de Tipos de Dados (uso de Wrappers)*

Agenda

- [CRONOGRAMA DA DISCIPLINA]


python


VIRTVS VNITA FORTIOR

O que é um paradigma de programação?

- Pode-se dizer que um paradigma expressa um padrão de pensamento de uma comunidade.
- Existem diferentes comunidades de programadores, bem como diferentes domínios de aplicação, é natural que haja diferentes paradigmas também dentro da área de programação.
- O uso de um paradigma de programação adequado ao domínio de aplicação permite que o programador desenvolva sistemas com maior produtividade, pois os códigos podem ser escritos de uma forma mais natural, mais legível e mais fácil de se manter ao longo da vida útil do sistema.

Resumindo...

De forma geral, um paradigma de programação proporciona e defini a visão que o programador possui sobre a estrutura e execução do programa, permitindo ou proibindo o uso de algumas técnicas de programação.

Exemplos

- Paradigma Imperativo (Ada, Cobol, C, Fortran, Pascal, Lua, Python, etc)
- Paradigma Orientado a Objetos (Java, Ruby, C++, C#, Python, etc)
- Paradigma Funcional (Haskell, Lisp, APL, etc)
- Paradigma Lógico (Prolog, Mercury, Visual Prolog, etc)

Paradigma Orientado a Objetos

- Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma. Para isso, a linguagem precisa atender a quatro tópicos bastante importantes:

Hello Python!

- Criada em 1991 por Guido van Rossum.
- Os objetivos do projeto da linguagem eram: produtividade e legibilidade, ou seja, Pyhton foi desenvolvida para criar código bom e fácil de manter de maneira rápida.
- Python suporta múltiplos paradigmas de programação.
- Python tem uma biblioteca padrão imensa, que contém classes, métodos e funções para realizar essencialmente qualquer tarefa, desde acesso a bancos de dados a interfaces gráficas com o usuário.
- É uma linguagem livre, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation; é multiplataforma.
- Possui baixo uso de caracteres especiais, o que torna a linguagem muito parecida com *pseudocódigo executável.*

O que vamos precisar?

- Uma implementação da Linguagem Python
  - http://www.python.org
  - Implementação pronta para baixar (Windows).
  - Linux: normalmente já vem com o python instalado.
- Um editor de textos
  - Qualquer editor simples.
  - Ambiente IDLE já inclui um editor
  - Pode ser utilizado também IDEs como Eclipse ou Netbeans ou outra com suporte a Python.

Variáveis

- Python é uma linguagem dinamicamente tipada. Isso quer dizer que não é necessário tipar as variáveis para usá-las, em outras palavras, não precisamos declarar as variáveis antes de utilizá-las.
- Para saber qual o tipo de determinado objeto usamos ***type*** *(nomedavariavel)*

```
>>> nome = 'Wendley'
>>> nome
'Wendley'
>>> type(nome)
<class 'str'>
```

```
>>> x = 2
>>> x
2
>>> type(x)
<class 'int'>
```

Expressões booleanas

- Também pode ser chamada de expressões lógicas
- Resultam em **verdadeiro (True)** ou **falso (False)**
- A expressão é avaliada da esquerda para direita
- Em alguns casos o resultado pode ser determinado sem avaliação da expressão inteira.
- Operadores mais utilizados:
  - Relacionais: **>, <, ==, !=, >=, <=**
  - Booleanas: **or, not, and**

Tipos de objetos

- Os principais tipos de objetos são inteiros, floats, strings, listas, tuplas e dicionários.
- **Strings:** são objetos que Python oferece para trabalhos com texto. Em Pyhton, as strings são sequências, permitindo um trabalho mais eficiente com elas do que em outras linguagens.
  - As Strings são endereçadas de tal forma que você consegue requisitar um valor qualquer dessa sequência e fazer operações com ele.
  - Os endereçamentos começam a ser contados de zero.
  - Sintaxe para requisitar o valor atribuído a um endereço X de uma sequência S é **S[X]**.

```
>>> palavra = 'python'
>>> palavra[2]
't'
```

Exercícios básicos

- Último índice da sequência:

  >>> palavra[-1]

- Solicitar um intervalo de uma sequência:

  >>> palavra [3:6]

- Podemos omitir o endereço da direita, se ele for o último da sequência:

  >>> palavra [3:]

- Tamanho da sequência:

  >>> len(palavra)

- Concatenação de strings:

  >>> palavra+palavra

  >>> “campus ”+’Sobral’

Tipos de objetos

- **Lista:** conjunto ordenado de valores, onde cada valor é identificado por um índice. Os valores que compõem uma lista são chamados elementos. Os elementos de uma lista podem possuir qualquer tipo.
- A lista é definida por [ ] e os seus elementos são separados por vírgula.
- Os elementos podem ser de qualquer tipo: inteiros, strings, listas, números complexos, etc.

```
>>> lista = [1,2,4,5,6]
>>> [10, 'alo', 2.2, [1,19]]
```

- Os elementos da lista podem ser acessados por índices.
  - O 1º elemento tem índice 0
  - O último elemento tem índice -1
- As listas constituem o tipo de agregação de dados mais versátil e comum de Python.
- Podem ser usadas para implementar estruturas de dados mais complexas como matrizes e árvores.

Listas x Strings

- Lista é uma sequência genérica, a strings é uma sequência de caracteres.
- **Strings** são composta de caracteres e imutáveis, **listas** são compostas de elementos de qualquer tipo e mutáveis.
- Os elementos da lista podem ser alterados individualmente. Strings funcionam assim?? (Não, professor!)

Listas

- O operador + pode ser usado para concatenação de listas e * para repetição:

```
>>> notas = [10]*3
>>> notas + ['ap1','ap2','ap3']
>>> ['Andre'] + notas
```

- Deletando elementos:

```
>>> lista2 = [1,'a','bc',[19,7+8j]]
>>> del lista2[1]
>>> lista2
[1, 'bc', [19, (7+8j)]]
del lista2[2][1]
>>> lista2
[1, 'bc', [19]]
```

Listas

- Operador **in**: permite saber se um elemento pertence a sequência. Funciona para strings também.

```
>>> lista = [1, 'a', 'bc']
>>> 1 in lista
 True
```

- **len(lista)** retorna o número de elementos da lista.
- **min(lista)** retorna o menor elemento da lista.
- **max(lista)** retorna o maior elemento da lista.

Quer converter uma string em lista?

- A função **list** faz isso!

```
>>> lista = list('alo')
>>> lista
['a', 'l', 'o']
```

- Quer fazer o processo inverso? Use o método **join**

```
p = ['di','ver','ti','do']
>>> ''.join(p)
'divertido‘
>>> prof = ''.join(p)
```

Tuplas

- **Tuplas** são similares as listas exceto por serem imutáveis. Tupla é uma lista de valores separados por vírgulas.
- Um valor do tipo tupla é uma série de valores separados por vírgulas e entre ( ).

```
>>> tupla = ('a', 'b', 'c', 'd', 'e')
>>> tupla [0]
‘a
```

- A função **tuple** constrói uma trupla a partir de qualquer sequência:

```
>>> tuple("abcd")
('a', 'b', 'c', 'd')
```

Tipos de objetos

- Os tipos compostos aprendidos até aqui – strings, listas, tuplas – utilizam índices inteiros.
- **Dicionários:** São similares aos tipos compostos, porém podem usar qualquer tipo imutável como índices, como strings. O sistema de endereçamento dos dicionários é por chaves. Cada chave tem um valor atribuído.
  - Primeiro passo: Criar um dicionário. Forma mais fácil é defini-lo como vazio.

  >>> tradutor = {}

  - Segundo passo: Adicionar elementos

  >>> tradutor ['one'] = 'um'

  >>> tradutor ['two'] = 'dois'

  - Outra forma de criar um dicionário é fornecendo uma lista de pares chave-valor:

  >>> tradutor = {'one': 'um', 'two' : 'dois'}

IF, ELIF, ELSE

- Instrução condicional simples:

```
if x > 0:
    print ("x é positivo")
```

- Instrução condicional composta:

```
if x > 0:
    print ("x é positivo")
else:
    print ("x é negativo")
```

- Instrução condicional encadeada:

```
if x < y:
    print (x, "é menor que", y)
elif x > y:
    print (x, "é maior que", y)
else:
    print (x, "e", y, "são iguais")
```

FOR

- Com o for podemos percorrer qualquer sequência em Python (lista, tupla, string) ou dicionários.
  - Exemplo 1:

```
for numero in range(20):
    if numero % 2 == 0:
        print (numero)
```

  - Exemplo 2:

```
a = ['Joao','Rafael','Douglas','Gustavo']
for i in range(len(a)):
    print(i,a[i])
```

  - Exemplo 3:

```
S = ‘elefante’
for i in S:
    print(i)
```

While

- Exemplo 1:

```
>>> b = 0
>>> while b < 5:
        print(b)
        b = b+1
```

- Exemplo 2:

```
>>> b = 1
>>> while b < 6:
        print('%i dolares = %.2f reais'%(b,b*3.5))
        b = b+1
```

Tratamento de erros

- Sintaxe:

```
try:
    primeiro tenta fazer isso
    e tudo que estiver neste alinhamento
    neste bloco de código
except:
   se qualquer linha dentro do bloco try
   der erro
   serão executadas essas linhas
```

Tratamento de erros

- Exemplo:

```
x = int(input('Digite um número inteiro: '))  #recebe um valor do usuário
while x < 3:
    try:
        print (1./x)
    except:
        print ('Não é possível dividir 1 por ', x)
    x+=1
```

Funções

- Funções podem ser definidas:
  - Por você (programador)
  - Em módulos da biblioteca padrão
  - Por default: são as funções embutidas (built-in)
- Esboço de uma função:

```
def nome (arg1, arg2, arg3,...):
    comando
    comando
    ...
    ...
    return <expressão>
```

Onde:

- *nome* é o nome da função
- *Arg1, arg2, arg3* são especificações dos argumentos.
- Uma função pode ter 0 a n argumentos
- *Comandos* são as instruções que serão executadas quando a função é invocada.

Funções

- Para indicar o valor a ser devolvido como o resultado da função, é o usado o comando **return.**
- Ao encontrar o return, a função termina imediatamente e o controle do programa volta ao ponto onde a função foi chamada .
- Se uma função chega a seu fim sem nenhum valor de retorno ter sido especificado, o valor de retorno é **None.**
  - None não pertence a nenhum tipo.
  - É um valor nulo.

Argumentos *Default*

- Podemos dar valores *default* a argumentos.
- Formato:

  def *nome* (*arg1: default, ..., argn: default*)
- Se apenas alguns argumentos têm default, esses devem ser os últimos para evitar ambiguidade na passagem dos parâmetros.
- Exemplo:

```
def hello(nome,saudacao='Olá',pontuacao=‘ :)'):
    return saudacao+', '+nome+pontuacao


>>> hello(“Mariana")
'Olá, Mariana :)‘
```

- Podemos especificar valores para os argumentos declarados como default, se quisermos.

```
>>> hello(“Mariana“,”Oi”,(“!”))
'Oi, Mariana!'
```

Documentação de Funções

- É sempre bom descrever a função. Ao invés de usar comentários para isso, é mais vantajoso usar **docstrings.**
  - Uma constante string escrita logo após o comando def.
- O acesso a documentação é feito através do interpretador, usando a notação:

  nomedafunçao.__**doc**__
- Exemplo:

```
def fat(n):
    "Retorna o fatorial de n"
    for i in range(n-1,1,-1): n*=i
    return n
>>> fat(6)
720
>>> print(fat.__doc__)
Retorna o fatorial de n
```

Importando módulos

- Módulos podem conter não apenas funções, mas também constantes e variáveis.
- Muitas funções importantes são definidas em módulos da biblioteca padrão.
- Para usar os elementos de um módulo, é usado o comando **import**.
- Sintaxe:
  - **import** *modulo*
    - importa todos os elementos do módulo
  - **from** *modulo* **import** *nomefunçao,...,nomefunção*
    - Importa apenas funções especificas do módulo
  - **from** *modulo* **import** *
    - Importa todos os elementos do módulo
    - todos os elementos têm que ser citados precedidos pelo *nome do módulo.*

Importando módulos

- Exemplo 1:

```
import math
>>> x = sin(90)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    x = sin(90)
NameError: name 'sin' is not defined
>>> x = math.sin(90)
0.893996663600557 9
>>> math.pi
3.141592653589793
```

Importando módulos

- Exemplo 2:

```
from math import *
>>> x = sin(90)
>>> x
0.893996663600579
>>> print(pi)
3.141592653589793
```

Revise os conhecimentos aprendidos hoje.