Estácio

ARA0066 - PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO EM PYTHON

PROFESSOR: LUCAS SAMPAIO LEITE

Motivação...

- Computadores apenas executam tarefas que lhe são passadas por meio dos programas que são construídos a partir das linguagens de programação. Dessa forma é importante a definição de que tipo de dados estão sendo trabalhados e sua localização sejam indicados de forma correta.
- Suponha que um programador precisa decidir que tipo de dado uma variável que armazena valores de giroscópio, acelerômetro e barômetro que equipa um drone e sabendo que que os valores devem ser os mais precisos possíveis. Que parâmetros ele utilizará para tomar essa decisão? Qual a implicação de uma escolha equivocada?

Motivação…

O acidente com o Ariane 5 no vôo inaugural | Um erro simples que custou $500 Milhões de dólares

Outras falhas históricas: https://www.bbc.com/portuguese/noticias/2015/05/150513_vert_fut_bug_digital_ml

Sumário

- Variáveis
- Vinculação
  - Vinculação de tipos
    - Vinculação de tipos Estática x Dinâmica
    - Variáveis sem tipo
- Vinculação de Armazenamento
  - Verificações
  - Verificação de tipos
- Tipos de dados

Memória x Variável

Memória

- A memória é uma sequência de células;
- Cada célula possui um endereço e um status atual:
  - alocada ou desalocada.
- Cada célula alocada possui um conteúdo, que pode ser um valor armazenado ou pode ser um valor indefinido.

Memória x Variável

Variável

- Uma variável de programa é uma abstração de uma célula ou de um conjunto de células de memória;
- São utilizadas para armazenar valores e representar objetos do mundo real;
- O conteúdo de uma variável pode ser inspecionado e atualizado sempre que necessário.

Variáveis (atributos)

- Uma variável apresenta seis atributos básicos:
  1. Nome ou identificador;
  2. Endereço (l-value);
  3. Tipo;
  4. Valor (r-value);
  5. Escopo;
  6. Tempo de vida.

Variáveis (atributos)

1. **Nome ou identificador:**
   - Sequência de caracteres utilizada para identificar a variável;
   - Questões Relevantes:
     - Qual o tamanho máximo de um nome?
     - Quais caracteres podem ser utilizados em nomes?
     - Os nomes fazem distinção entre maiúsculas e minúsculas?

Variáveis (atributos)

1. **Nome ou identificador:**
   - Tamanho do nome
     - Se muito pequeno, não são conotativos;
     - Exemplos de linguagens:
       - FORTRAN I: máximo de 6;
       - COBOL: máximo de 30;
       - FORTRAN 90 e ANSI C: máximo de 31;
       - Ada e Java: sem limite;
       - C++: sem limite, mas implementadores geralmente impõem limites.

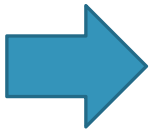
Variáveis (atributos)

1. **Nome ou identificador:**
   - Distinção entre maiúsculas e minúsculas:
     - Exemplos: nome, Nome, NOME;
     - Nomes em C, em C++, em Python e em Java fazem distinção entre maiúsculo e minúsculo.
   - Desvantagens:
     - Legibilidade:
       - Nomes semelhantes representam entidades diferentes.
   - Palavra reservada:
     - Uma palavra reservada é uma palavra especial que não pode ser usada como nome.

Variáveis (atributos)

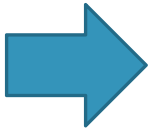
1. **Nome ou identificador:**

```
ex.py > ...
1  x = int(5.0)
2  print(type(x))
3  int = 5
4  print(5)
```



```
lucas@lucas-Inspiron-5548:~
x.py
<class 'int'>
5
```

```
ex.py
1  class = int(5.0)
2
```



```
lucas@lucas-Inspiron-5548:~/Drop
x.py
  File "/home/lucas/Dropbox/IFPE
    class = int(5.0)
            ^
SyntaxError: invalid syntax
```

Variáveis (atributos)

1. **Nome ou identificador:**
   - Palavras reservadas em python:

and	as	assert	break
class	continue	def	del
elif	else	except	False
finally	for	from	global
if	import	in	is
lambda	None	nonlocal	not
or	pass	raise	return
True	try	while	with
yield			

Variáveis (atributos)

2. **Endereço (l-value):**
   - É o endereço da posição de memória associada com a variável;
   - Questões Relevantes:
     - O mesmo nome pode ser associado a diferentes endereços em **diferentes partes de um mesmo programa**;

```
1  program enderecos1;
2  var num : integer;
3  procedure teste;
4      var num : real;
5      begin
6          write('Digite um numero real: ');
7          readln(num);
8      end;
9      begin {programa principal}
10         writeln('Digite um numero inteiro: ');
11         readln(num);
12         teste;
13     end.
```

Variáveis (atributos)

2. **Endereço (l-value):**
   - É o endereço da posição de memória associada com a variável;
   - Questões Relevantes:
     - O mesmo nome pode ser associado a diferentes endereços em diferentes partes de um mesmo programa;
     - O mesmo nome pode ser associado a diferentes endereços em **diferentes momentos de um mesmo programa**;

```
1  program enderecos2;
2  var num : integer;
3  function fatorial (n : integer);
4      begin
5          if (n=0) or (n=1)
6              then fatorial := 1
7              else fatorial := n*fatorial(n-1);
8      end;
9      begin
10         readln(num);
11         writeln(fatorial(num));
12     end.
```

Variáveis (atributos)

2. **Endereço (l-value):**
   - Aliases:
     - Quando mais de um nome de variável pode ser utilizado para acessar uma única localização de memória, os nomes são chamados de aliases.
     - Problemas de legibilidade.

ex.py > ...

```python
a=[1,2,3]
b=a
print(b is a)
b[0]=10
print(a)
print(b)
```

```c
int *x, valor, y;
valor = 35;
x = &valor; //atr
```

```c
typedef struct pessoa { //definição da struct pessoa
    char nome[50], endereco[50];
    int idade, id;
    char sexo;
}TPessoa;
```

Variáveis (atributos)

Os tipos de dados padrão do Python são:

- Inteiro (int)
- Ponto Flutuante ou Decimal (float)
- **Tipo** Complexo (complex)
- String (str)
- Boolean (bool)
- List (list)
- Tuple.
- Dictionary (dic)

3. **Tipo:**

- É o tipo dos dados que serão armazenados na variável, que define os valores possíveis e as operações permitidas;
- O tipo determina:
  - A faixa de valores que a variável pode ter:
    - INTEGER em algumas versões do FORTRAN (-32.768 a 32.767).
  - Conjunto de operações definidas para os valores do tipo:
    - Para tipos numéricos: Soma, Subtração, Multiplicação, Divisão, Exponenciação...
    - Para Strings em Java, por exemplo: Concatenação, Substituição.

Variáveis (atributos)

4. **Valor (r-value):**

- É o conteúdo armazenado na posição de memória associada com a variável;
- Utilizamos o conceito de célula de memória abstrata;
- Também chamado de VALOR-R, pois é o que se exige quando a variável é utilizada do lado direito de uma atribuição;
- Para acessar o VALOR-R, o VALOR-L é determinado antes.

Variáveis (atributos)

5. **Escopo:**

- Bloco do programa em que o nome da variável é conhecido;

6. **Tempo de vida:**
   - Tempo durante o qual a variável está vinculada a uma posição de memória.
   - O tempo de vida (lifetime) de uma variável é o tempo durante o qual ela está vinculada a uma localização de memória específica;
   - O tempo de vida de um variável é o intervalo de tempo decorrente entre a sua criação (alocação) e a sua deleção (desalocação).

Variáveis (atributos)

- Escopo local:
  - Bloco de código em que uma variável pode ser acessada;
  - Uma variável criada dentro de uma função pertence ao escopo local dessa função;
  - Logo, só pode ser usada dentro dessa função.
- Escopo global:
  - Variável criada no corpo principal de um arquivo python;
  - Podem ser acessadas tanto no escopo global quanto local
  - Uma variável criada fora de uma função pode ser acessada dentro e fora de uma função.

Variáveis (atr

```python
varGlobal = 'Sou uma variavel global \o/'
def showMeTheCode():
    varLocal = 'Sou uma variável local o/'
    print (varLocal)
    print (varGlobal)
showMeTheCode()
print (varGlobal)
print (varLocal)
```

```
Sou uma variável local o/
Sou uma variavel global \o/
Sou uma variavel global \o/
Traceback (most recent call last):
  File "c:\Users\Lucas Sampaio\Dropbox\IFPE\CursoFerias\project\main
.py", line 8, in <module>
    print (varLocal)
NameError: name 'varLocal' is not defined. Did you mean: 'varGlobal'
?
```

```c
#include <stdio.h>

int variavelGlobal = 10; // Variável global

void funcao1()
{
    int variavelLocal1 = 20; // Variável local à função funcao1
    printf("VariavelGlobal dentro de funcao1: %d\n", variavelGlobal);
    printf("VariavelLocal1 dentro de funcao1: %d\n", variavelLocal1);
}

void funcao2()
{
    int variavelLocal2 = 30; // Variável local à função funcao2
    printf("VariavelGlobal dentro de funcao2: %d\n", variavelGlobal);
    printf("VariavelLocal2 dentro de funcao2: %d\n", variavelLocal2);
}
```

```c
    int variavelLocal3 = 40; // Variável local à função main
    printf("VariavelGlobal dentro de main: %d\n", variavelGlobal);
    printf("VariavelLocal3 dentro de main: %d\n", variavelLocal3);

    funcao1();
    funcao2();

    return 0;
}
```

Variáveis (atributos)

```c
#include <stdio.h>
void func1(){
    int b;
    b = -100;
    printf("Valor de b dentro da função func1: %d\n", b);
}
void func2(){
   int b;
   b = -200;
   printf("Valor de b dentro da função func2: %d\n", b);
}
```

```c
int main(){
    int b;

    b = 10;
    printf("Valor de b: %d\n", b);
    b = 20;
    func1();
    printf("Valor de b: %d\n", b);
    b = 30;
    func2();
    printf("Valor de b: %d\n", b);
    return 0;
}
```

O que será impresso???

Variáveis (atributos)

```c
#include <stdio.h>
void func1(){
    int b;
    b = -100;
    printf("Valor de b dentro da função func1: %d\n", b);
}
void func2(){
   int b;
   b = -200;
   printf("Valor de b dentro da função func2: %d\n", b);
}
```

```
Valor de b: 10
Valor de b dentro da função func1: -100
Valor de b: 20
Valor de b dentro da função func2: -200
Valor de b: 30
```

```c
int main(){
    int b;

    b = 10;
    printf("Valor de b: %d\n", b);
    b = 20;
    func1();
    printf("Valor de b: %d\n", b);
    b = 30;
    func2();
    printf("Valor de b: %d\n", b);
    return 0;
}
```

Variáveis (vinculação)

- Antes que uma variável possa ser referenciada em um programa, ela deve ser vinculada a um tipo de dados;
- Vinculação Estática:
  - Ocorre **antes** do tempo de execução de um programa, e;
  - Permanece **inalterada** durante a execução do programa.
- Vinculação Dinâmica:
  - Ocorre **durante** a execução de um programa, ou;
  - Pode ser **modificada** durante a execução do programa.

Variáveis (vinculação de tipos)

- Antes de ser referenciada num programa, uma variável precisa ser vinculada a um tipo de dados;
- Duas questões são importantes:
  - A maneira como o tipo é especificado.
    - Declaração implícita ou explícita.
  - Quando a vinculação ocorre.
    - Estática ou Dinâmica.

Variáveis (vinculação de tipos)

- Declaração explícita:
  - Instrução em um programa que lista nomes de variáveis e especifica que elas são de um tipo particular.
- Declaração implícita
  - Um meio de associar variáveis a tipos por convenções em vez de instruções;
  - A primeira ocorrência de um nome de variável corresponde à sua declaração implícita.
- FORTRAN, PL/I, BASIC e Perl dispõem de declarações implícitas.
  - Vantagem:
    - Capacidade de escrita (conveniências para programadores).
  - Desvantagem:
    - Legibilidade (erros tipográficos passam na compilação).

Variáveis (vinculação de tipos estática)

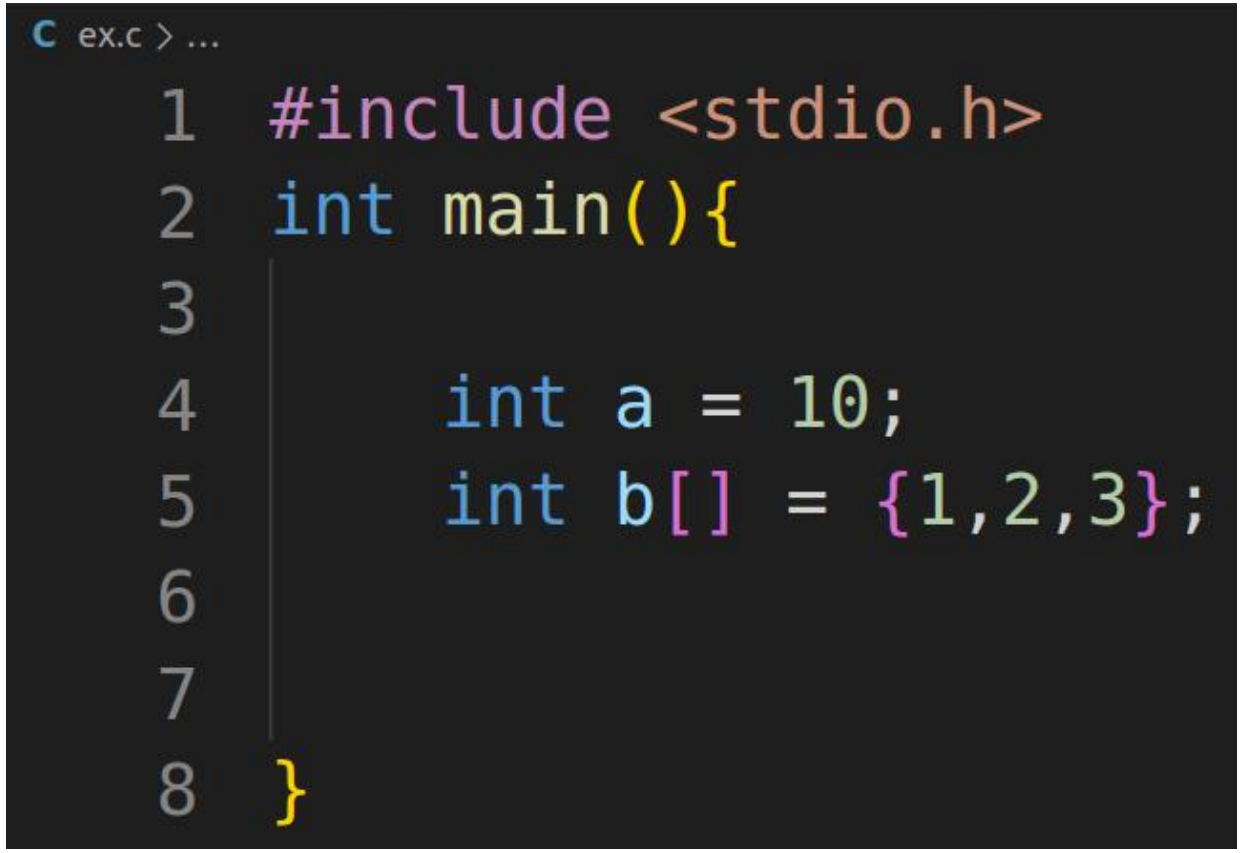
- Declaração de Variáveis:
  - Associa, estaticamente, uma variável a um tipo de dados;
    - Ocorre em tempo de compilação;
  - **Declaração explícita:**
  - É uma instrução em um programa que lista nomes de variáveis e especifica que elas são de um determinado tipo.
    - Exemplos: Pascal, C, C++, Java;
    - Pascal (sempre no início de um bloco).

```
1  var i : integer;
2  opcao : char;
3  x, y : real;
```

  - Em algumas linguagens, as declarações de variáveis podem ocorrer em qualquer posição dentro de um bloco.
    - C (início do bloco)
    - C++ e Java (qualquer posição do bloco).

```
1  int i;
2  char opcao;
3  float x, y;
```

Variáveis (vinculação de tipos estática)



Variáveis (vinculação de tipos estática)

- Declaração de Variáveis:
  - **Declaração implícita:**
    - A primeira ocorrência de um nome de variável em um programa constitui sua declaração implícita;
    - É um meio de associar variáveis a tipos de dados através de convenções padrão sem utilizar instruções de declaração;
      - Exemplos: ML, Perl, Fortran, Basic.
      - Exemplo no Fortran ->

```
1  I,J,K,L,M,N -> INTEGER
2  Demais letras -> REAL
```

    - Conveniência para o programador;
    - Dificulta as checagens do compilador;
    - Problemas de legibilidade.

Variáveis (vinculação de tipos estática)

- Declaração de Variáveis:
  - **Declaração implícita:**
    - Associar nomes iniciados com determinados caracteres especiais a tipos de dados específicos.
      - Exemlos: Perl (qualquer posição do bloco)
      - Exemplo no Fortran:

```
1  $num = 7
2      # Iniciando com $ Tipo simples (escalar)
3  @nome = ("Paulo", "Leandro", "Severina")
4      # Iniciando com @ Tipo array
```

    - Conveniência para o programador;
    - Dificulta as checagens do compilador;
    - Problemas de legibilidade.

Variáveis (vinculação de tipos dinâmica)

- O tipo de uma variável não é especificado por nenhuma instrução de declaração;
  - O tipo vai depender da variável ou expressão do lado direito de uma atribuição.
- A variável é vinculada a um tipo de dados quando lhe é atribuído um valor em uma instrução de atribuição;
- Presente, frequente, em linguagens interpretadas ao invés de compiladas.
- Exemplos: PHP, APL, SNOBOL4, LISP, SMALLTALK.
  - PHP:

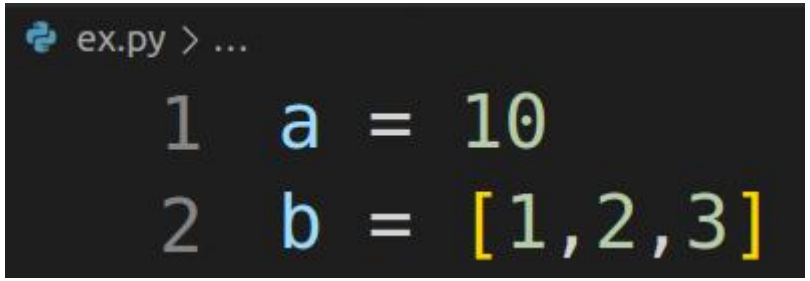
```
1 $teste = "0"; //$teste e um string (ASCII 48)
2 $teste = $teste + 1; //$teste agora e inteiro (1)
3 $teste = $teste + 1.3; //$teste agora e real (2.3)
```

  - APL:

```
1 LIST <- 47 //LIST e uma variavel do tipo inteiro
2 LIST <- 10.2 5.1 0.0 //LIST e um array de reais
```

Variáveis (vinculação de tipos dinâmica)

Python possui vinculação dinâmica!!!!



Variáveis (vinculação de tipos dinâmica)

- Vantagem:
  - Flexibilidade.
- Desvantagens:
  - Diminui a capacidade do compilador de detecção de erros;
  - Alto custo de implementação:
    - Checagem de tipos em tempo de execução;
    - Memória de tamanho variável.

Variáveis (sem tipo)

- São variáveis que não estão associadas a um tipo de dados específico;
- Podem armazenar valores de qualquer tipo de dados;
- Não fazem checagem de tipos.
- Exemplo: JavaScript (em qualquer posição do bloco)

```
1 var i;
2 i = 10;
3 i = "dez";
```

Tipo de dado

- Em programação, um tipo de dado é uma classificação que define o tipo de valor que uma variável pode armazenar em um programa de computador.
- Os tipos de dados são usados para definir o espaço de armazenamento necessário para uma variável, as operações que podem ser realizadas em cima dela e a forma como os dados serão interpretados pelo computador.

Tipo de dado

- Os tipos de dados são fundamentais para a declaração e manipulação de variáveis em linguagens de programação.
- Eles podem incluir tipos básicos, como inteiros (por exemplo, int), números de ponto flutuante (por exemplo, float), caracteres (por exemplo, char), booleanos (por exemplo, bool), entre outros.
- Além disso, algumas linguagens de programação permitem a definição de tipos de dados personalizados, conhecidos como tipos de dados compostos ou referenciados, como arrays, estruturas, classes, entre outros.

Tipo de dado

- Os tipos de dados são importantes porque determinam o espaço de memória necessário para armazenar os valores, a faixa de valores que podem ser representados, as operações que podem ser executadas nesses valores e como os dados são interpretados e processados pelo computador.
- O uso correto de tipos de dados é fundamental para garantir a corretude, eficiência e segurança dos programas de computador.

Tipo de dado

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis digitos de precisão
double	8	Dez digitos de precisão
long double	10	Dez digitos de precisão

E em Python?

O maior número inteiro representado no Python não tem um limite específico, pois depende da arquitetura do sistema em que o Python está sendo executado.

No entanto, em sistemas baseados na arquitetura de 64 bits, o maior número inteiro que pode ser representado é 2^63 - 1, que é igual a 9.223.372.036.854.775.807.

Tipo de dados primitivos

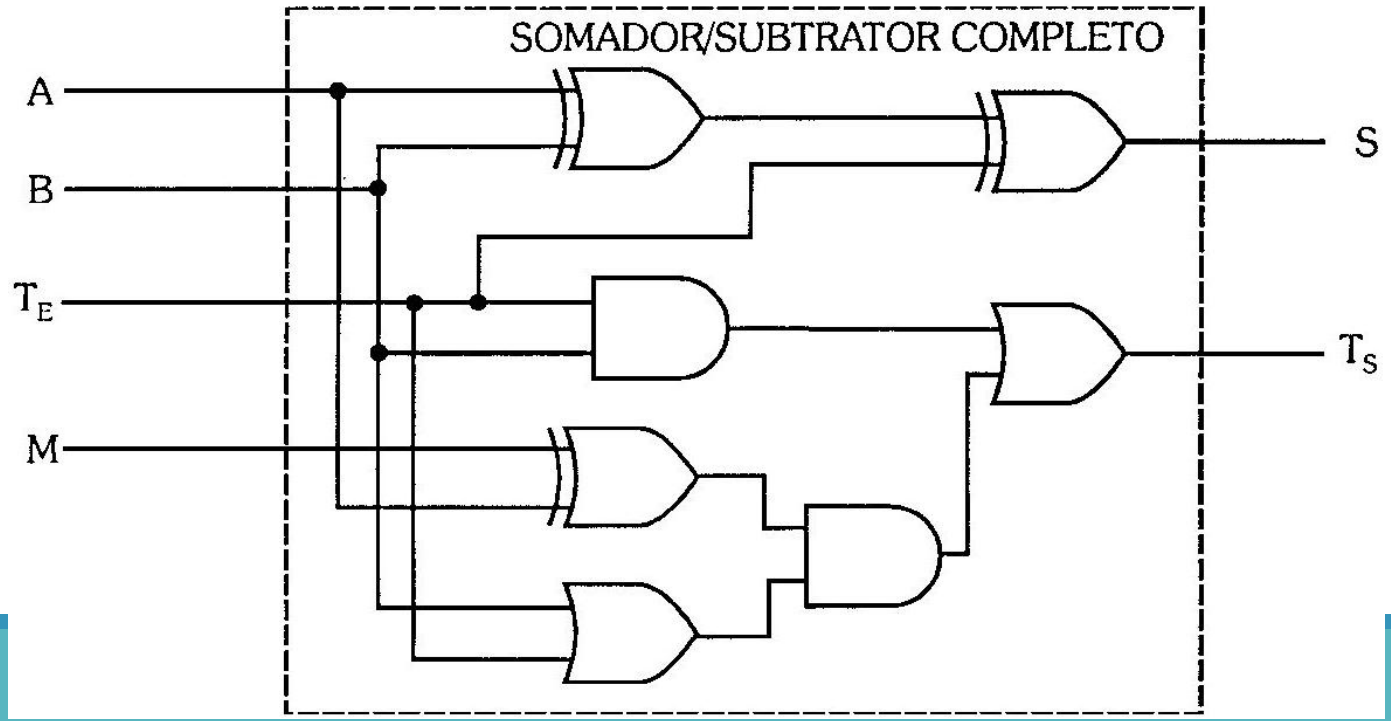
- Tipos de dados primitivos são tipos de dados básicos oferecidos por uma linguagem de programação que não são compostos de outros tipos de dados. Eles são usados para representar os valores fundamentais ou primitivos em um programa de computador.
- Os tipos de dados primitivos variam de uma linguagem de programação para outra, mas geralmente incluem os seguintes tipos:
  - Inteiro (int): Representa números inteiros, positivos ou negativos, sem casas decimais. Pode ter diferentes tamanhos, como inteiros de 32 bits, 64 bits, etc., dependendo da linguagem de programação.
  - Ponto flutuante (float ou double): Representa números com casas decimais. O tipo float é normalmente de precisão simples, enquanto o tipo double é normalmente de precisão dupla, sendo mais preciso, mas ocupando mais espaço na memória.

Tipo de dados primitivos

- Tipos de dados primitivos são tipos de dados básicos oferecidos por uma linguagem de programação que não são compostos de outros tipos de dados. Eles são usados para representar os valores fundamentais ou primitivos em um programa de computador.
- Os tipos de dados primitivos variam de uma linguagem de programação para outra, mas geralmente incluem os seguintes tipos:
  - Caractere (char): Representa um único caractere, como uma letra, número ou símbolo.
  - Booleano (bool): Representa um valor verdadeiro (true) ou falso (false), frequentemente usado para expressar condições lógicas.
  - Outros tipos primitivos específicos da linguagem: Algumas linguagens de programação têm tipos primitivos adicionais, como byte (representando um valor inteiro de 8 bits), short (representando um inteiro curto), long (representando um inteiro longo), entre outros.

Tipo de dados primitivos

- Os tipos de dados primitivos são usados para armazenar valores simples em variáveis e são geralmente suportados diretamente pelo hardware do computador, o que os torna eficientes em termos de uso de memória e desempenho.
- Eles são a base para a construção de tipos de dados mais complexos e estruturas de dados em programas de computador.



Tipo de dados primitivos (ponto flutuante)

- Modelam os números reais, mas são aproximações;
- Linguagens para fins científicos suportam pelo menos dois tipos ponto flutuante float e double;
- IEEE Floating Point Standard 754:
  - Precisão simples:

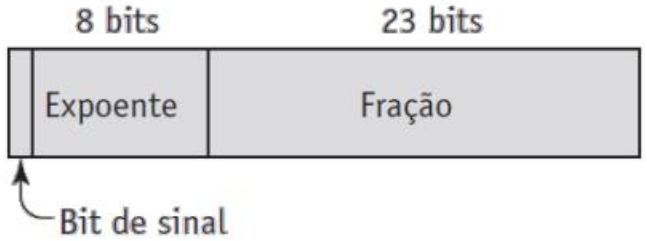


Figura extraída de Sebesta (2019, p. 272).

  - Precisão dupla:

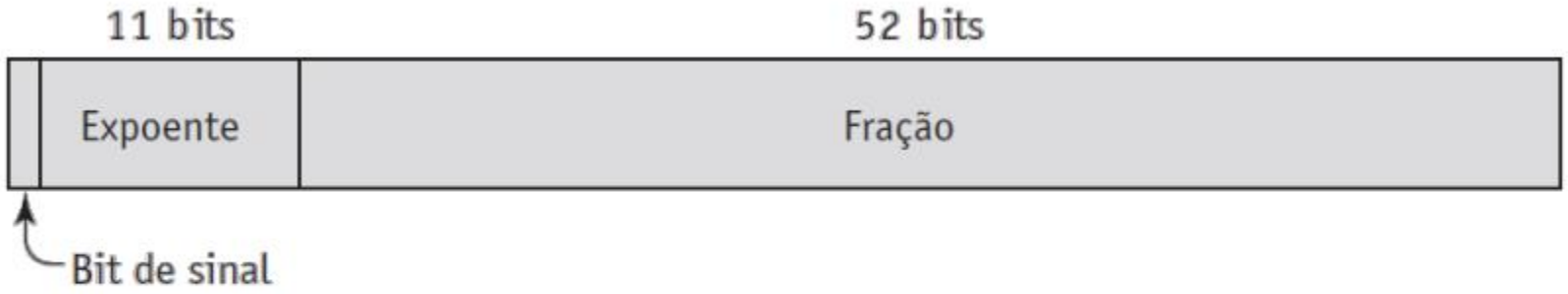


Figura extraída de Sebesta (2019, p. 272).

Tipo de dados primitivos (booleano)

- Mais simples de todos;
- Faixa de valores
  - Dois elementos, um para “‘true” e um para “false”.
- Pode ser implementado como bits, mas geralmente são como bytes;
- Vantagem:
  - Legibilidade.
- Expressões numéricas podem ser usadas como condicionais. Se diferente de 0 verdadeiro, e se igual a 0 falso.

Tipo de dados primitivos (booleano)

- Em C, o tipo de dado booleano não é nativamente suportado como um tipo de dado primitivo. Em vez disso, os valores booleanos são representados usando inteiros, onde zero (0) representa falso (false) e qualquer valor diferente de zero representa verdadeiro (true).
- Geralmente, é comum usar a biblioteca stdbool.h para definir os valores booleanos true e false.

```c
#include <stdbool.h> // Incluir a biblioteca stdbool.h

int main() {
    bool meuBooleano = true;

    if (meuBooleano) { // Verifica se o valor de meuBooleano é verdadeiro (true)
        printf("O booleano é verdadeiro (true)\n");
    } else {
        printf("O booleano é falso (false)\n");
    }

    return 0;
}
```

Tipo de dados primitivos (booleano)

```c
#define BOOL int // Definir o tipo BOOL como um inteiro
#define TRUE 1   // Definir o valor TRUE como 1
#define FALSE 0  // Definir o valor FALSE como 0

int main() {
    BOOL meuBooleano = TRUE;

    if (meuBooleano) {
        printf("O booleano é verdadeiro (TRUE)\n");
    } else {
        printf("O booleano é falso (FALSE)\n");
    }

    return 0;
}
```

Tipo de dados primitivos (booleano)

- Em Python, os booleanos são representados pelos valores True e False, que são palavras-chave reservadas da linguagem. Aqui está um exemplo simples de como você pode usar booleanos em Python:

```
meu_booleano = True

if meu_booleano:  # Verifica se o valor de meu_booleano é True
    print("O booleano é verdadeiro (True)")
else:
    print("O booleano é falso (False)")
```

Tipo de dados primitivos (caractere)

- Armazenados como codificações numéricas;
- O código mais usado: ASCII (7 bits);
  - Exemplo linguagem C.
- Uma alternativa, codificação de 16 bits: Unicode
  - Inclui caracteres da maioria das linguagens naturais;
  - Usado em Java;
  - C# e JavaScript também suportam Unicode.
  - ASCII é subconjunto de Unicode.

Tipo de dados primitivos (caractere)

- As funções ord() e chr() são funções embutidas do Python que permitem converter entre valores inteiros representando caracteres e os próprios caracteres.

- A função ord() retorna o valor Unicode de um caractere especificado como argumento. Já a função chr() retorna o caractere representado por um valor Unicode especificado como argumento.

```
# Obtém o valor Unicode do caractere 'A'
valor_unicode = ord('A')
print(valor_unicode)  # Saída: 65

# Obtém o valor Unicode do caractere 'ô'
valor_unicode = ord('ô')
print(valor_unicode)  # Saída: 244


# Exemplo com chr()

# Obtém o caractere representado pelo valor Unicode 65
caractere = chr(65)
print(caractere)  # Saída: 'A'

# Obtém o caractere representado pelo valor Unicode 244
caractere = chr(244)
print(caractere)  # Saída: 'ô'
```

Cadeia de caracteres (Strings)

- Strings são sequências de caracteres. Um caractere é um símbolo, como uma letra, número, espaço em branco ou um símbolo especial, que pode ser representado em um computador.
- Uma string é, portanto, uma sequência ou coleção de caracteres.
- Em outras palavras, uma string é uma representação de texto em um programa de computador.
- Operações típicas:
  - Atribuição e cópia;
  - Comparação (==, !=, etc.);
  - Concatenação;
  - Referências a subcadeias;

Cadeia de caracteres (Strings)

- Uso nas linguagens de programação:
  - C e C++
    - Não primitivo;
    - Usam vetores char e uma biblioteca de funções que oferecem operações (string.h).
  - SNOBOL4 (uma linguagem de manipulação de cadeias)
    - Primitivo;
  - Java
    - Objeto através da classe String.
  - Python
    - Objeto da classe str

Cadeia de caracteres (Strings)

- Em Python, uma string é uma sequência imutável de caracteres, e a linguagem fornece várias operações embutidas para manipular strings:
  - Concatenação de strings: É possível concatenar duas ou mais strings usando o operador de adição (+). Por exemplo:

```
s1 = "Olá"
s2 = "Mundo"
s3 = s1 + " " + s2
print(s3)  # Saída: "Olá Mundo"
```

Cadeia de caracteres (Strings)

- Em Python, uma string é uma sequência imutável de caracteres, e a linguagem fornece várias operações embutidas para manipular strings:
  - Multiplicação de strings: É possível repetir uma string múltiplas vezes usando o operador de multiplicação (*). Por exemplo:

```
s = "Python"
s_repetida = s * 3
print(s_repetida)  # Saída: "PythonPythonPython"
```

Cadeia de caracteres (Strings)

- Em Python, uma string é uma sequência imutável de caracteres, e a linguagem fornece várias operações embutidas para manipular strings:
  - Indexação de strings: É possível acessar caracteres individuais em uma string usando a notação de índice, onde o índice começa em 0 para o primeiro caractere. Por exemplo:

```
s = "Hello"
print(s[0])  # Saída: "H"
print(s[1])  # Saída: "e"
print(s[-1])  # Saída: "o"
```

Cadeia de caracteres (Strings)

- Em Python, uma string é uma sequência imutável de caracteres, e a linguagem fornece várias operações embutidas para manipular strings:
  - Fatias de strings: É possível obter uma parte específica de uma string usando a notação de fatia (slicing), que permite acessar uma sequência de caracteres. Por exemplo:

```
s = "Hello"
print(s[1:4])
```

Cadeia de caracteres (Strings)

- Em Python, uma string é uma sequência imutável de caracteres, e a linguagem fornece várias operações embutidas para manipular strings:
  - Métodos de strings: Python também fornece uma variedade de métodos embutidos para manipulação de strings, como len(), lower(), upper(), strip(), replace(), split(), entre outros. Por exemplo:

```
s = " Olá, Mundo! "
print(len(s))  # Saída: 15 (comprimento da string)
print(s.lower())  # Saída: " olá, mundo! " (conversã
print(s.upper())  # Saída: " OLÁ, MUNDO! " (conversã
print(s.strip())  # Saída: "Olá, Mundo!" (remoção de
print(s.replace("Mundo", "Python"))  # Saída: " Olá,
print(s.split(","))  # Saída: [' Olá', ' Mundo! '] (
```

Vetores e matrizes

- Vetores e matrizes são estruturas de dados usadas na programação para armazenar e manipular coleções de elementos.
- Vetor: É uma sequência ordenada de elementos do mesmo tipo de dado, representados como uma única dimensão de valores. Um vetor pode ser unidimensional, como uma lista de números inteiros [1, 2, 3, 4], ou multidimensional, como uma lista de pontos tridimensionais [(1, 2, 3), (4, 5, 6), (7, 8, 9)].
- Matriz: É uma coleção bidimensional de elementos dispostos em linhas e colunas. É uma tabela retangular de elementos, onde cada elemento é identificado por sua posição na linha e na coluna. Uma matriz pode ter várias linhas e colunas, como uma tabela de valores numéricos, por exemplo.

Vetores e matrizes

- As matrizes são frequentemente usadas para representar dados tabulares, como imagens, planilhas de dados, mapas de pixels, entre outros.
- Os vetores e matrizes são usados em muitos campos da ciência da computação e matemática, como processamento de imagem, aprendizado de máquina, análise de dados, gráficos computacionais, simulações, entre outros.
- A manipulação de vetores e matrizes é uma parte fundamental em muitos algoritmos e programas de computador, e é suportada por várias linguagens de programação, incluindo Python (implementada a partir de listas), Java, C++, entre outras.

Vetores e matrizes (C)

```c
#include <stdio.h>

int main() {
    // Declarando e inicializando um vetor de inteiros com 5 elementos
    int vetor[5] = {1, 2, 3, 4, 5};

    // Acessando e imprimindo o valor de um elemento específico do vetor
    printf("Valor do segundo elemento do vetor: %d\n", vetor[1]);

    return 0;
}
```

Vetores e matrizes (C)

```c
#include <stdio.h>

int main() {
    // Declarando e inicializando uma matriz de inteiros com 3 linhas e 3 colunas
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Acessando e imprimindo o valor de um elemento específico da matriz
    printf("Valor do elemento na segunda linha e segunda coluna: %d\n", matriz[1][1]);

    return 0;
}
```

Vetores e matrizes (Python)

- Em Python, vetores e matrizes podem ser implementados usando listas, que são estruturas de dados dinâmicas e flexíveis que podem armazenar uma coleção de itens, incluindo números, strings, objetos ou até mesmo outras listas.

```python
# Vetor de números inteiros
vetor_inteiros = [1, 2, 3, 4, 5]

# Vetor de números reais
vetor_reais = [1.2, 2.3, 3.4, 4.5]

# Vetor de strings
vetor_strings = ["hello", "world", "python"]

# Acessar elementos do vetor
print(vetor_inteiros[0])  # Saída: 1
print(vetor_reais[1])  # Saída: 2.3
print(vetor_strings[2])  # Saída: "python"

# Modificar elementos do vetor
vetor_inteiros[3] = 99
print(vetor_inteiros)  # Saída: [1, 2, 3, 99, 5]

# Tamanho do vetor
tamanho = len(vetor_strings)
print(tamanho)  # Saída: 3
```

Vetores e matrizes (Python)

```python
# Matriz de números inteiros
matriz_inteiros = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Matriz de números reais
matriz_reais = [[1.1, 2.2, 3.3], [4.4, 5.5, 6.6], [7.7, 8.8, 9.9]]

# Matriz de strings
matriz_strings = [["hello", "world"], ["python", "is", "awesome"]]

# Acessar elementos da matriz
print(matriz_inteiros[0][0])  # Saída: 1
print(matriz_reais[1][2])  # Saída: 6.6
print(matriz_strings[1][1])  # Saída: "is"

# Modificar elementos da matriz
matriz_inteiros[1][1] = 99
print(matriz_inteiros)  # Saída: [[1, 2, 3], [4, 99, 6], [7, 8, 9]]
```

Exercícios

- Implemente as seguintes questões utilizando a linguagem Python e uma segunda linguagem (Ex: C, C++, Java, etc)

1. Exercício de Inteiros e Operações Matemáticas: Escreva um programa que peça ao usuário para inserir dois números inteiros e, em seguida, exiba a soma, subtração, multiplicação e divisão desses números.
2. Exercício de Strings e Manipulação de Texto: Escreva um programa que peça ao usuário para inserir seu nome completo e, em seguida, exiba o nome completo em letras maiúsculas, letras minúsculas e o número de caracteres no nome.
3. Exercício de Listas e Acesso a Elementos: Escreva um programa que crie uma lista (ou um vetor) de números inteiros e, em seguida, exiba o primeiro elemento, o último elemento e o terceiro elemento dessa lista.
4. Exercício Matrizes: Escreva um programa que crie uma matriz 5x5, preencha-a com valores aleatórios e, em seguida, imprima a sua diagonal principal.

Dúvidas???