UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

R para Ciências da Vida (BCM13065)
Aula 5

PPGBCM - UFRGS

Diego Bonatto

2024/2

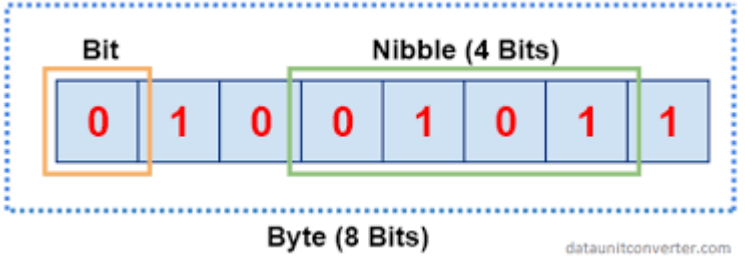
Tipos de dados e variáveis em linguagens de programação

- Cada linguagem possui tipos de dados específicos (ex.: caractere, inteiro, flutuante) para armazenar informações e realizar operações.
- A memória alocada varia conforme o tipo de dado (ex.: em C, caractere = 1 byte, inteiro = 2 ou 4 bytes).
- Em R: variáveis são atribuídas a objetos, não a tipos de dados fixos.

Definições básicas – bits e bytes

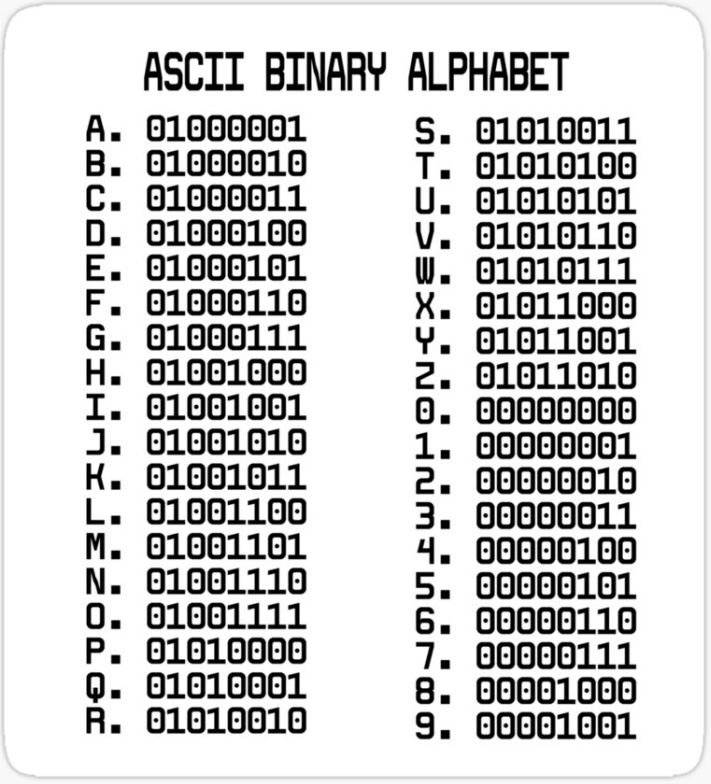
Memory Capacity Conversion Chart

Term (Abbreviation)	Approximate Size
Byte (B)	8 bits
Kilobyte (KB)	1024 bytes / 10^3 bytes
Megabyte (MB)	1024 KB / 10^6 bytes
Gigabyte (GB)	1024 MB / 10^9 bytes
Terabyte (TB)	1024 GB / 10^{12} bytes
Petabyte (PB)	1024 TB / 10^{15} bytes
Exabyte (EB)	1024 PB / 10^{18} bytes
Zettabyte (ZB)	1024 EB / 10^{21} bytes
Yottabyte (YB)	1024 ZB / 10^{24} bytes



- Um número binário é uma forma de representar números usando apenas dois símbolos: 0 e 1. Este sistema é chamado de "sistema numérico binário" ou "sistema numérico de base 2".
- Pense nisso como um interruptor de luz – está ligado (1) ou desligado (0). Esta é a ideia básica por trás dos números binários.

Definições básicas – bits e bytes



ASCII BINARY ALPHABET

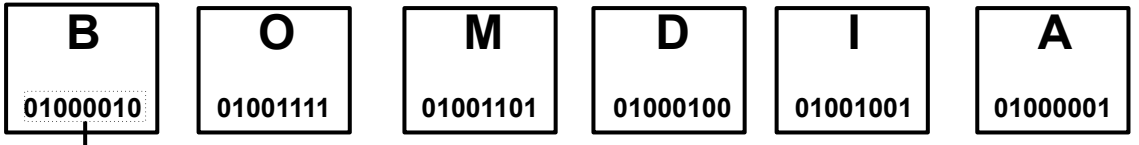
A.	01000001	S.	01010011
B.	01000010	T.	01010100
C.	01000011	U.	01010101
D.	01000100	V.	01010110
E.	01000101	W.	01010111
F.	01000110	X.	01011000
G.	01000111	Y.	01011001
H.	01001000	Z.	01011010
I.	01001001	0.	00000000
J.	01001010	1.	00000001
K.	01001011	2.	00000010
L.	01001100	3.	00000011
M.	01001101	4.	00000100
N.	01001110	5.	00000101
O.	01001111	6.	00000110
P.	01010000	7.	00000111
Q.	01010001	8.	00001000
R.	01010010	9.	00001001

Cada tipo de dado é uma sequência de bytes. Cada byte é uma sequência de 8 bits (dígitos binários) e portanto tem $2^8 = 256$ possíveis valores:

00000000, 00000001, 00000010, ... ,11111110, 11111111

Os bytes são convertidos para dados passíveis de leitura por humanos utilizando tabelas de conversão, como ASCII.

Definições básicas – bits e bytes



$0\times2^7 + 1\times2^6 + 0\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$0 + 64 + 0 + 0 + 0 + 0 + 2 + 0 = 66$

Tabela ASCII

A	chr(65)	01000001	Â³	chr(179)	10110011
B	chr(66)	01000010	Â´	chr(180)	10110100
C	chr(67)	01000011	Âµ	chr(181)	10110101
D	chr(68)	01000100	Â¶	chr(182)	10110110
E	chr(69)	01000101	Â·	chr(183)	10110111
F	chr(70)	01000110	Â¸	chr(184)	10111000
G	chr(71)	01000111	Â¹	chr(185)	10111001
H	chr(72)	01001000	Âº	chr(186)	10111010
I	chr(73)	01001001	Â»	chr(187)	10111011
J	chr(74)	01001010	Â¼	chr(188)	10111100
K	chr(75)	01001011	Â½	chr(189)	10111101

https://www.binaryhexconverter.com/binary-ascii-characters-table

Tipos de Dados e Alocação de Memória

- **Caractere (a, b, c...):** 1 byte
- **Número Inteiro (1, 2, 3...):** 1-4 bytes
- **Ponto Flutuante (1.1, 2.3, 5.6...):** 4-8 bytes
- **Longo (“Olá mundo”...):** 8 bytes

- A alocação de memória varia conforme o tipo de dado e a linguagem de programação.

Por que o tamanho dos dados importa?

- **Alocação de Memória:** Evitar estouros de memória e otimizar o uso de recursos.
- **Performance:** Dados menores são processados mais rapidamente.
- **Escolha de Algoritmos:** Diferentes algoritmos são mais eficientes para diferentes tamanhos de dados.
- **Transmissão de Dados:** Impacta a velocidade de transferência e o consumo de banda.
- **Armazenamento:** Determina a quantidade de espaço em disco necessária.

Tipos de dados em R

1. Vetores

- **Descrição**: Estrutura de dados unidimensional que armazena elementos do mesmo tipo (ex.: numérico, caractere).
- **Uso de RAM**:
  - **Eficiente**, já que só aceita elementos homogêneos.
  - Tamanho da memória é proporcional ao número de elementos e ao tipo de dado (inteiros usam menos memória que flutuantes).
- **Tempo de Processamento**:
  - **Muito rápido**, especialmente para operações elementares ou funções vetorizadas.
- **Exemplo de Aplicação**: Somar um vetor de 1 milhão de números.

Tipos de dados em R

2. Arrays

- **Descrição**: Extensão de vetores para mais dimensões.
- **Uso de RAM**:
  - **Moderado a alto**, pois ocupa mais memória para representar múltiplas dimensões.
  - Aloca a memória para todas as posições, mesmo que estejam vazias.
- **Tempo de Processamento**:
  - **Depende da dimensão**, mas é geralmente eficiente para operações matemáticas vetorizadas.
- **Exemplo de Aplicação**: Análises de múltiplas variáveis (ex.: séries temporais multidimensionais).

Tipos de dados em R

3. Matrizes

- **Descrição**: Representação bidimensional de vetores com elementos homogêneos.
- **Uso de RAM**:
  - Semelhante a arrays, com **alocação proporcional ao tamanho** e tipo dos dados.
  - Menos flexível que dataframes (só aceita um tipo de dado).
- **Tempo de Processamento**:
  - **Rápido**, especialmente para cálculos matriciais otimizados por bibliotecas como BLAS/LAPACK.
- **Exemplo de Aplicação**: Álgebra linear e modelos estatísticos.

Tipos de dados em R

4. Dataframes

- **Descrição**: Estrutura bidimensional que suporta diferentes tipos de dados em colunas.
- **Uso de RAM**:
  - **Moderado a alto**, porque cada coluna pode ter seu próprio tipo e o overhead para organização é maior.
  - Tende a consumir mais memória em comparação a matrizes para o mesmo número de elementos.
- **Tempo de Processamento**:
  - **Mais lento** do que matrizes para operações de grande escala, devido à flexibilidade e checagem de tipos.
  - Operações com pacotes otimizados (ex.: **data.table**, **dplyr**) são mais rápidas.
- **Exemplo de Aplicação**: Manipulação e análise de grandes conjuntos de dados tabulares.

Tipos de dados em R

5. Listas

- **Descrição**: Estrutura que pode armazenar diferentes tipos de objetos, incluindo outros vetores, matrizes, ou até listas.
- **Uso de RAM**:
  - **Alto**, pois precisa armazenar referências para cada elemento e seus metadados.
  - Tamanho depende do número e complexidade dos objetos armazenados.
- **Tempo de Processamento**:
  - **Mais lento** para acessar ou modificar elementos, devido à flexibilidade.
  - Operações específicas (ex.: indexação) podem ser otimizadas.
- **Exemplo de Aplicação**: Armazenar resultados de modelos ou dados heterogêneos.

Tipos de dados e uso de RAM/processamento em R

Comparação Geral

Estrutura	Uso de RAM	Tempo de Processamento	Nota
Vetor	Baixo	Muito rápido	Ideal para operações homogêneas simples.
Array	Moderado	Rápido	Eficiente para dados multidimensionais homogêneos.
Matriz	Moderado	Muito rápido	Excelente para cálculos matemáticos e lineares.
Dataframe	Alto	Moderado	Versátil, mas consome mais memória devido à heterogeneidade.
Lista	Muito alto	Mais lento	Alta flexibilidade, mas com custo significativo em RAM e processamento.

Tipos de dados e uso de RAM/processamento em R

- Vetores, matrizes e arrays são mais eficientes em termos de RAM e processamento, mas são limitados a dados homogêneos.

- Dataframes e listas oferecem flexibilidade para manipular dados heterogêneos, mas a um custo maior de desempenho e uso de memória.

- Escolha ideal: Depende do tipo de análise e tamanho do conjunto de dados. Para eficiência, prefira vetores/matrizes quando possível; para flexibilidade, use dataframes/listas.

Tipos de análises de dados biológicos e alocação

Comparação Geral

Análise	Uso de RAM	Tempo de Processamento
Alinhamento de Sequências	Moderado (MB-GB)	Variável (segundos a horas)
Modelagem de Proteínas	Muito alto (GB-TB)	Longo (horas a dias)