Data Types: Complete Properties & Methods

String Properties & Methods:

Properties:

1. .length - Returns number of characters

String text = 'Hello ';

print(text.length); // Output: 10

print(''.length); // Output: 0

2. .isEmpty - Returns true if string is empty

String empty = '';

String notEmpty = 'Hello';

print(empty.isEmpty); // Output: true

print(notEmpty.isEmpty); // Output: false

3. .isNotEmpty - Returns true if string is not empty

String name = 'John';

String nothing = '';

print(name.isNotEmpty); // Output: true

print(nothing.isNotEmpty); // Output: false

Creation Methods:

1. String.fromCharCode(codeUnit) - Create from ASCII code

String charA = String.fromCharCode(65);

print(charA); // Output: 'A' (ASCII 65 = 'A')

String charZ = String.fromCharCode(90);

print(charZ); // Output: 'Z' (ASCII 90 = 'Z')

2. String.fromCharCodes(codeUnits) - Create from list of codes

```
List<int> codes = [72, 69, 76, 76, 79]; // H E L L O
String hello = String.fromCharCodes(codes);
print(hello); // Output: 'HELLO'
String  = String.fromCharCodes([68, 97, 114, 116]);
print(); // Output: ''
```

3. String.fromEnvironment(name, defaultValue) - Get from environment

```
// This reads from environment variables
// Usually used in compile-time constants
const String apiKey = String.fromEnvironment('API_KEY', defaultValue: 'default');
print(apiKey); // Output: 'default' (or actual value if set
```

Checking Methods:

1. .contains(pattern) - Checks if contains substring

```
String sentence = ' is awesome';
print(sentence.contains('is'));      // Output: true
print(sentence.contains('IS'));      // Output: false (case-sensitive)
print(sentence.contains(RegExp(r'[A-Z]'))); // Output: true (has uppercase)
```

2. .startsWith(pattern) - Checks if starts with

```
String url = 'https://example.com';
print(url.startsWith('https'));  // Output: true
print(url.startsWith('http'));   // Output: true
print(url.startsWith('www'));    // Output: false
```

3. .endsWith(pattern) - Checks if ends with

```
String fileName = 'document.pdf';
print(fileName.endsWith('.pdf'));  // Output: true
```

```
print(fileName.endsWith('.doc'));  // Output: false
```

4. .compareTo(other) - Compares two strings lexicographically

```
print('apple'.compareTo('banana'));  // Output: -1 (apple < banana)
print('banana'.compareTo('apple'));  // Output: 1 (banana > apple)
print('apple'.compareTo('apple'));   // Output: 0 (equal)
print('Apple'.compareTo('apple'));   // Output: -1 (uppercase < lowercase)
```

5. .codeUnitAt(index) - Returns UTF-16 code unit at index

```
String text = '';
print(text.codeUnitAt(0));  // Output: 68 (D)
print(text.codeUnitAt(1));  // Output: 97 (a)
print(text.codeUnitAt(3));  // Output: 116 (t)
```

Transformation Methods:

1. .toLowerCase() - Converts to lowercase

```
String mixed = 'Hello  World';
print(mixed.toLowerCase()); // Output: 'hello  world'
```

2. .toUpperCase() - Converts to uppercase

```
String text = 'hello world';
print(text.toUpperCase()); // Output: 'HELLO WORLD'
```

3. .trim() - Removes leading and trailing whitespace

```
String padded = '  Hello World  ';
print(padded.trim());       // Output: 'Hello World'
print('  '.trim());         // Output: '' (empty string)
print(' hello '.trim());    // Output: 'hello'
```

4. .trimLeft() - Removes leading whitespace only

```
String text = '   Left trim   ';
print(text.trimLeft()); // Output: 'Left trim   '
```

5. .trimRight() - Removes trailing whitespace only

```
String text = '   Right trim   ';
print(text.trimRight()); // Output: '   Right trim'
```

6. .padLeft(width, [pad]) - Pads left side with character

```
String number = '42';
print(number.padLeft(5, '0'));   // Output: '00042'
print('Hi'.padLeft(4));          // Output: '  Hi' (default pad: space)
print('*'.padLeft(3, '-'));      // Output: '--*'
```

7. .padRight(width, [pad]) - Pads right side with character

```
String name = 'John';
print(name.padRight(8, '.'));    // Output: 'John....'
print(''.padRight(6));       // Output: '  '
```

Searching Methods:

1. .indexOf(pattern, [start]) - Returns first occurrence index

```
String text = 'hello world hello';
print(text.indexOf('world'));    // Output: 6
print(text.indexOf('hello'));    // Output: 0
print(text.indexOf('hello', 1)); // Output: 12 (start searching from index 1)
print(text.indexOf('goodbye'));  // Output: -1 (not found)
```

2. .lastIndexOf(pattern, [start]) - Returns last occurrence index

```
String text = 'hello world hello world';
print(text.lastIndexOf('world'));    // Output: 18
print(text.lastIndexOf('hello'));    // Output: 12
print(text.lastIndexOf('goodbye'));  // Output: -1
```

3. .substring(start, [end]) - Extracts substring

```
String text = ' Programming';
print(text.substring(0, 4));     // Output: ''
print(text.substring(5));        // Output: 'Programming' (from index 5 to end)
print(text.substring(0, text.indexOf(' '))); // Output: ''
```

4. .split(pattern) - Splits string into list

```
String csv = 'apple,banana,orange';
print(csv.split(','));           // Output: ['apple', 'banana', 'orange']
String sentence = 'Hello  World';
print(sentence.split(' '));      // Output: ['Hello', '', 'World']
// Split by multiple characters
String data = 'a,b,c-d|e';
print(data.split(RegExp(r'[,|-]'))); // Output: ['a', 'b', 'c', 'd', 'e']
```

Replacement Methods:

1. .replaceAll(from, to) - Replaces all occurrences

```
String text = 'I love Java and Java is great';
print(text.replaceAll('Java', ''));
// Output: 'I love  and  is great'
String spaces = 'a  b    c';
print(spaces.replaceAll(RegExp(r'\s+'), ' '));
// Output: 'a b c'
```

2. .replaceFirst(from, to) - Replaces first occurrence only

```
String text = 'apple apple apple';
print(text.replaceFirst('apple', 'banana'));
// Output: 'banana apple apple'
print(text.replaceFirst('orange', 'banana'));
// Output: 'apple apple apple' (no change, not found)
```

3. .replaceRange(start, end, new) - Replaces range

```
String text = 'Hello World';
print(text.replaceRange(6, 11, ''));
// Output: 'Hello '
print(text.replaceRange(0, 5, 'Hi'));
// Output: 'Hi World'
```

Utility Methods:

1. .toString() - Returns string representation

```
String text = 'Hello';
print(text.toString()); // Output: 'Hello' (same string)
```

2. .hashCode - Returns hash code

```
String text1 = 'Hello';
String text2 = 'Hello';
String text3 = 'World';
print(text1.hashCode); // Output: 177593 (example value)
print(text2.hashCode); // Output: 177593 (same as text1)
print(text3.hashCode); // Output: 397349 (different)
```

3. .runtimeType - Returns type at runtime

String text = '';

print(text.runtimeType); // Output: String

---

int Properties & Methods:

Properties:

1. .isEven - Returns true if number is even

print(10.isEven); // Output: true

print(7.isEven); // Output: false

print(0.isEven); // Output: true

print((-4).isEven); // Output: true

2. .isOdd - Returns true if number is odd

print(7.isOdd); // Output: true

print(10.isOdd); // Output: false

print(1.isOdd); // Output: true

print((-3).isOdd); // Output: true

3. .isNegative - Returns true if number is negative

print((-5).isNegative); // Output: true

print(0.isNegative); // Output: false

print(10.isNegative); // Output: false

4. .sign - Returns -1, 0, or 1

print((-10).sign); // Output: -1 (negative)

print(0.sign); // Output: 0 (zero)

print(42.sign); // Output: 1 (positive)

5. .bitLength - Bits needed to represent the number

```
print(1.bitLength);     // Output: 1 (binary: 1)
print(2.bitLength);     // Output: 2 (binary: 10)
print(7.bitLength);     // Output: 3 (binary: 111)
print(255.bitLength);   // Output: 8 (binary: 11111111)
print(256.bitLength);   // Output: 9 (binary: 100000000)
```

6. .isFinite - Always true for int

```
print(10.isFinite);   // Output: true
print((-5).isFinite); // Output: true
print(0.isFinite);    // Output: true
```

7. .isInfinite - Always false for int

```
print(100.isInfinite); // Output: false
print(0.isInfinite);   // Output: false
```

8. .isNaN - Always false for int

```
print(42.isNaN);   // Output: false
print((-1).isNaN); // Output: false
```

Mathematical Methods:

1. .abs() - Returns absolute value

```
print((-10).abs()); // Output: 10
print(10.abs());    // Output: 10
print(0.abs());     // Output: 0
```

2. .clamp(lower, upper) - Clamps to range

```
print(15.clamp(10, 20));  // Output: 15 (within range)
print(5.clamp(10, 20));   // Output: 10 (too small)
```

```
print(25.clamp(10, 20));  // Output: 20 (too large)
print((-5).clamp(0, 100)); // Output: 0
```

3. .gcd(other) - Greatest common divisor

```
print(12.gcd(18));    // Output: 6
print(15.gcd(25));    // Output: 5
print(17.gcd(31));    // Output: 1 (coprime)
print(0.gcd(5));      // Output: 5
```

4. .modInverse(modulus) - Modular inverse

```
print(7.modInverse(11)); // Output: 8 (7*8 mod 11 = 1)
print(3.modInverse(10)); // Output: 7 (3*7 mod 10 = 1)
```

5. .modPow(exponent, modulus) - Modular exponentiation

```
print(2.modPow(3, 5));    // Output: 3 (2³ mod 5 = 8 mod 5 = 3)
print(5.modPow(3, 13));   // Output: 8 (5³ mod 13 = 125 mod 13 = 8)
```

6. .remainder(other) - Remainder of division

```
print(10.remainder(3));   // Output: 1
print(10.remainder(4));   // Output: 2
print((-10).remainder(3)); // Output: -1
```

Bitwise & Conversion Methods:

1. .toRadixString(radix) - Converts to base string

```
print(10.toRadixString(2));   // Output: '1010' (binary)
print(255.toRadixString(16)); // Output: 'ff' (hexadecimal)
print(10.toRadixString(8));   // Output: '12' (octal)
print(42.toRadixString(10));  // Output: '42' (decimal)
```

2. .toSigned(width) - Returns signed representation

```
print(255.toSigned(8));   // Output: -1 (8-bit signed)
print(128.toSigned(8));   // Output: -128
```

3. .toUnsigned(width) - Returns unsigned representation

```
print((-1).toUnsigned(8)); // Output: 255 (8-bit unsigned)
print((-128).toUnsigned(8)); // Output: 128
```

Parsing Methods (static):

1. int.parse(string, [radix]) - Parse string to int

```
print(int.parse('42'));        // Output: 42
print(int.parse('FF', radix: 16)); // Output: 255
print(int.parse('1010', radix: 2)); // Output: 10
// int.parse('abc'); // Throws FormatException
```

2. int.tryParse(string) - Safe parsing (returns null on error)

```
print(int.tryParse('42'));      // Output: 42
print(int.tryParse('abc'));     // Output: null
print(int.tryParse('123.45'));  // Output: null
print(int.tryParse('  123  ')); // Output: 123
```

Constants:

```
print(int.maxFinite);  // Output: 9223372036854775807
print(int.minFinite);  // Output: -9223372036854775808
```

---

double Properties & Methods:

Properties:

1. .isFinite - Returns true if number is finite

```
print(3.14.isFinite);     // Output: true
print(double.infinity.isFinite); // Output: false
print(double.nan.isFinite);      // Output: false
```

2. .isInfinite - Returns true if number is infinite

```
print(double.infinity.isInfinite);  // Output: true
print(double.negativeInfinity.isInfinite); // Output: true
print(3.14.isInfinite);            // Output: false
```

3. .isNaN - Returns true if number is Not-a-Number

```
print(double.nan.isNaN);      // Output: true
print(3.14.isNaN);           // Output: false
print(double.infinity.isNaN); // Output: false
```

4. .isNegative - Returns true if number is negative

```
print((-3.14).isNegative); // Output: true
print(3.14.isNegative);    // Output: false
print(0.0.isNegative);     // Output: false
```

5. .sign - Returns -1.0, 0.0, or 1.0

```
print((-5.5).sign); // Output: -1.0
print(0.0.sign);    // Output: 0.0
print(3.14.sign);   // Output: 1.0
```

Mathematical Methods:

1. .abs() - Absolute value

```
print((-3.14).abs()); // Output: 3.14
print(3.14.abs());    // Output: 3.14
```

2. .clamp(lower, upper) - Clamps to range

```
print(15.5.clamp(10.0, 20.0));  // Output: 15.5
print(5.5.clamp(10.0, 20.0));   // Output: 10.0
print(25.5.clamp(10.0, 20.0));  // Output: 20.0
```

3. .ceil() - Smallest integer >= value

```
print(3.14.ceil());    // Output: 4
print(3.0.ceil());     // Output: 3
print((-3.14).ceil()); // Output: -3
```

4. .floor() - Largest integer <= value

```
print(3.14.floor());    // Output: 3
print(3.0.floor());     // Output: 3
print((-3.14).floor()); // Output: -4
```

5. .round() - Nearest integer

```
print(3.14.round());    // Output: 3
print(3.5.round());     // Output: 4
print(3.49.round());    // Output: 3
print((-3.5).round());  // Output: -4
```

6. .truncate() - Removes decimal part

```
print(3.14.truncate());    // Output: 3
print(3.99.truncate());    // Output: 3
```

print((-3.14).truncate()); // Output: -3

7. .ceilToDouble() - Ceil as double

print(3.14.ceilToDouble()); // Output: 4.0

print((-3.14).ceilToDouble()); // Output: -3.0

8. .floorToDouble() - Floor as double

print(3.99.floorToDouble()); // Output: 3.0

print((-3.14).floorToDouble()); // Output: -4.0

9. .roundToDouble() - Round as double

print(3.14.roundToDouble()); // Output: 3.0

print(3.5.roundToDouble()); // Output: 4.0

10. .truncateToDouble() - Truncate as double

print(3.99.truncateToDouble()); // Output: 3.0

print((-3.14).truncateToDouble()); // Output: -3.0

Formatting Methods:

1. .toString() - String representation

print(3.14.toString()); // Output: '3.14'

print((-5.5).toString()); // Output: '-5.5'

2. .toStringAsExponential([fractionDigits]) - Exponential notation

print(1234.567.toStringAsExponential()); // Output: '1.234567e+3'

print(1234.567.toStringAsExponential(2)); // Output: '1.23e+3'

print(0.000123.toStringAsExponential(3)); // Output: '1.230e-4'

3. .toStringAsFixed(fractionDigits) - Fixed decimal places

```
print(3.14159.toStringAsFixed(2)); // Output: '3.14'
print(3.14159.toStringAsFixed(0)); // Output: '3'
print(9.9.toStringAsFixed(3));     // Output: '9.900'
```

4. .toStringAsPrecision(precision) - Specific precision

```
print(3.14159.toStringAsPrecision(3)); // Output: '3.14'
print(123.456.toStringAsPrecision(5)); // Output: '123.46'
print(123.456.toStringAsPrecision(3)); // Output: '123'
```

Parsing Methods (static):

1. double.parse(string) - Parse string to double

```
print(double.parse('3.14'));      // Output: 3.14
print(double.parse('42'));        // Output: 42.0
print(double.parse('  1.23  '));  // Output: 1.23
// double.parse('abc'); // Throws FormatException
```

2. double.tryParse(string) - Safe parsing

```
print(double.tryParse('3.14'));     // Output: 3.14
print(double.tryParse('abc'));      // Output: null
print(double.tryParse('123.45.67')); // Output: null
```

Constants:

```
print(double.infinity);         // Output: Infinity
print(double.negativeInfinity);  // Output: -Infinity
print(double.nan);              // Output: NaN
print(double.maxFinite);        // Output: 1.7976931348623157e+308
print(double.minPositive);      // Output: 5e-324
```

এই comprehensive documentation-এ প্রতিটি property এবং method-এর practical example দেওয়া হয়েছে, যাতে আপনি সহজেই বুঝতে পারেন কোন method কী করে এবং তার output কী হবে।