Compare By Codepoint

for Stage 1

Mathieu Hofman · Agoric

Mark S. Miller · Agoric

Christopher Hiller · METAMASK

107th Plenary

April 2025

TC39

Reminder: What are strings?

- Unicode ideal: sequence of unicode scalar values
  - Codepoints from 0 - 0x10FFFF
  - Minus surrogates (0xD800 - 0xDFFF)
- In JS: Sequence of 16-bit code units
  - UTF-16 encoding + lone surrogates
  - codepoints > 0xFFFF encoded as surrogate pairs
- For humans: sequence of Graphemes
  - visual “characters”
  - each a series of 1 or more Unicode scalar values

Example Graphemes and Codepoints

```
[...new Intl.Segmenter().segment('Ｅ𝗆o𝗃ｉ 🧑‍🎄')].map(({ segment }) => ({
  [segment]: [...segment].map(c => `0x${c.codePointAt(0).toString(16)}`),
}));
[
  { 'Ｅ': [ '0xff25' ] },
  { '𝗆': [ '0x1d5c6' ] },
  { o: [ '0x6f' ] },
  { '𝗃': [ '0x1d5c3' ] },
  { 'ｉ': [ '0xff49' ] },
  { ' ': [ '0x20' ] },
  { '🧑‍🎄': [ '0x1f9d1', '0x200d', '0x1f384' ] }, // 🧑🎄
]
```

Example JavaScript code units

```
'Ｅ𝗆o𝗃ｉ 🧑‍🎄'.split('').map(c => `0x${c.codePointAt(0).toString(16)}`);
```

```
[
  '0xff25',
  '0xd835', '0xddc6',
  '0x6f',
  '0xd835', '0xddc3',
  '0xff49',
  '0x20',
  '0xd83e', '0xddd1', '0x200d', '0xd83c', '0xdf84'
]
```

Where code units show up

- Index access
  - APIs dealing with offset, length, etc.
- Matching using RegExp
  - without the `u` or `v` flag
- Comparing strings
  - `arr.sort()`, `<`, `>`

Codepoint alternatives

- Index access
  - APIs dealing with offset, length, etc.

    ⇒ Iterator, `codePointAt`
- Matching using RegExp

  ⇒ the u or v flag

- Comparing strings
  - `arr.sort(), <, >`

    ⇒ ???

Codepoint aware comparators

❌`String.prototype.localeCompare()`
❌`get Intl.Collator.prototype.compare`

- Locale dependent
  - not stable over time
  - depends on the environment

- Meant for humans
  - groups confusables characters
  - collapses characters in same equivalence class

Locale comparators results

```
const arr = [
  '\u{ff42}', // Fullwidth Latin Small Letter B
  '\u{1d5ba}', // Mathematical Sans-Serif Small A
  '\u{63}', // Latin Small Letter C
];

[...arr].sort(); // [ 'c', '𝖺', 'ｂ' ]
[...arr].sort((a, b) => a.localeCompare(b)); // [ '𝖺', 'ｂ', 'c' ]
[...arr].sort(new Intl.Collator('zxx').compare); // [ '𝖺', 'ｂ', 'c' ]

new Intl.Collator("zxx").compare('\u0065\u0301','\u00e9') // 0
```

Portable comparator

- Need a comparator for data processing
- Compatible with other systems
    - Many languages represent strings as UTF-8
    - SQLite by default uses UTF-8 for strings
    - UTF-8 code units preserves Unicode codepoint sort order

⇒ **`String.codePointCompare`**

```
[...arr].sort(String.codePointCompare); // [ 'c', 'b', 'a' ]
```

Use case: Agoric custom collections

- Well defined sort order
  - specified order between data types
  - primitive types use their intrinsic order
  - insertion order for “object references”
  - other non comparable values are disallowed
- Different backing stores
  - Heap / Ephemeral: uses JS Map
  - Durable: uses SQLite DB

Questions? Stage 1?

Shim

```
function codePointCompare(left, right) {
  const leftIter = left[Symbol.iterator]();
  const rightIter = right[Symbol.iterator]();
  for (;;) {
    const { value: leftChar } = leftIter.next();
    const { value: rightChar } = rightIter.next();
    if (leftChar === undefined && rightChar === undefined) {
      return 0;
    } else if (leftChar === undefined) {
      // left is a prefix of right.
      return -1;
    } else if (rightChar === undefined) {
      // right is a prefix of left.
      return 1;
    }
    const leftCodepoint = leftChar.codePointAt(0);
    const rightCodepoint = rightChar.codePointAt(0);
    if (leftCodepoint < rightCodepoint) return -1;
    if (leftCodepoint > rightCodepoint) return 1;
  }
};
```