【文字列入門、基本的な使い方】

【文字列と null】1

サンプルコード

```
||null の印字表現は <null> です。
||文字列オブジェクトのインスタンスを作成するときに引数として null を指定すると、
||文字列には null の印字表現（つまり <null>）が格納されます
{let s:String = {String null}}

{value s}
```

実行結果

```
<null>
```

【文字列と null】2

サンプルコード

```
|| StringBuf利用時のサンプルです
{let sb:StringBuf = {StringBuf 99, null, "red balloons"}}

{value sb}
```

実行結果

```
99<null>red balloons
```

【文字クラス】

サンプルコード

```
||以下のコードでは、文字クラスを宣言して操作する方法を示しています。
{value
    ||変数を定義する
    ||CharClass は、非常に効率的に実装できる文字のセットです
    let vowels:CharClass = "AaEeIiOoUu"
    let sort-of-vowels:CharClass = {CharClass vowels, "Yy"}

    let vowels-count:int = 0
    let sort-of-vowels-count:int = 0
    let the-string:String = "Hello World!  Here comes that curly language!"

    || the-stringに対して処理を行う
    {for ch:char in the-string do
        ||member? メソッドは文字を引数として取り
        ||bool を返してその文字が文字クラスのメンバであるかどうかを示します
        {if {vowels.member? ch} then
            {inc vowels-count}
        }
        {if {sort-of-vowels.member? ch} then
            {inc sort-of-vowels-count}
        }
    }

    ||実行結果を確認する
    {paragraph
        I saw {value vowels-count} vowels and
        {value sort-of-vowels-count} sort-of vowels.
    }
}
```

実行結果

```
I saw 13 vowels and 14 sort-of vowels.
```

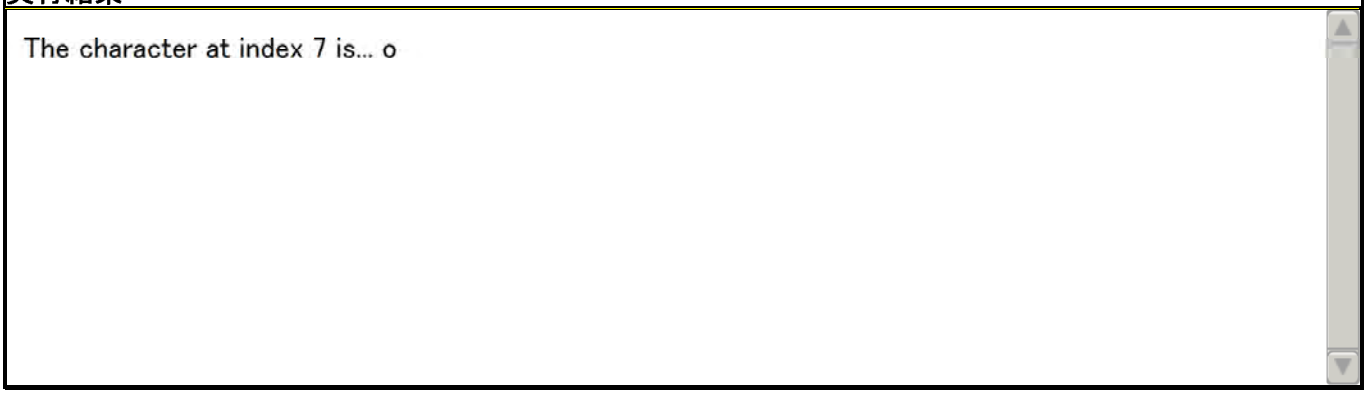
【個々の文字へのアクセス】1

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "Hello World"

    ||文字列内の個々の文字にアクセスするには get メソッドを使用します。
    ||文字のインデックスをメソッドに渡します。
    ||メソッドは受け取ったインデックスの文字を返します
    {text The character at index 7 is... {s.get 7}}
}
```

実行結果



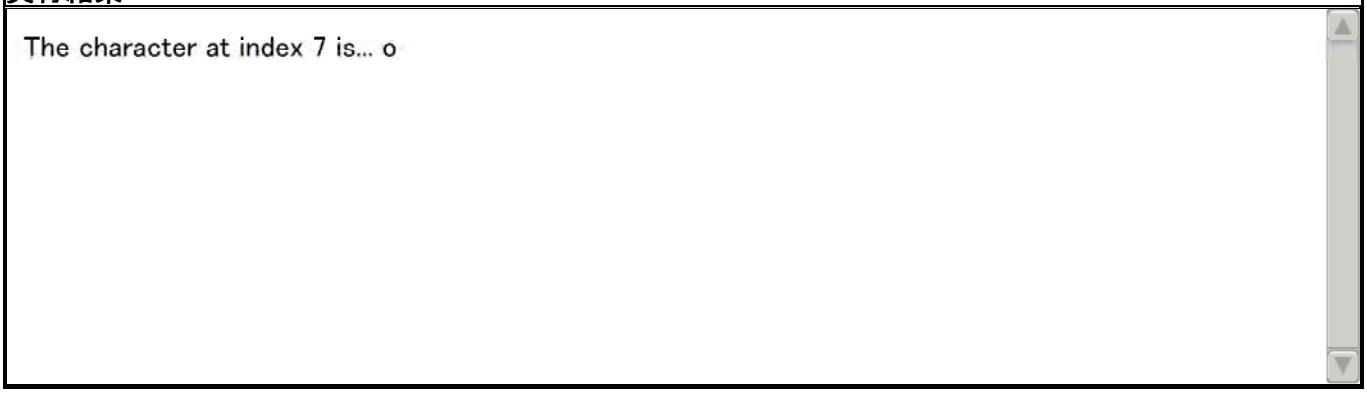
【個々の文字へのアクセス】2

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "Hello World"

    ||配列構文を使用して文字列内の文字にアクセスすることもできます
    {text The character at index 7 is... {value s[7]}}
}
```

実行結果



【部分文字列へのアクセス】1

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World"
    ||変数を定義する（部分文字列を返す）
    let s2:String = {s1.substr 6, 5}

    ||実行結果を確認してください
    {value s2}
}
```

実行結果

```
World
```

【部分文字列へのアクセス】2

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World"
    ||変数を定義する（文字列の指定位置と末尾の間のすべての文字を含む部分文字列を作成する）
    let s2:String = {s1.tail 6}

    ||実行結果を確認してください
    {value s2}
}
```

実行結果

```
World
```

【すべての要素へのアクセス】

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "Hello World"
    let sb:StringBuf = {StringBuf "Hi, "}

    ||コンテナ for ループを使用してアクセス可能です
    {for c:char in s do
        {sb.append c}
    }

    ||実行結果を確認してください
    {value sb}
}
```

実行結果

```
Hi, Hello World
```

【文字の追加】

サンプルコード

```
{value
    ||変数を定義する
    let sb:StringBuf = {StringBuf "Hell wor"}

    ||文字を追加する
    {sb.append 'l'}

    ||文字を追加する
    {sb.write-one 'd'}

    ||指定インデックスに文字を挿入する
    {sb.insert 'o', 4}

    ||指定インデックスに文字を挿入する
    {sb.splice " computer ", 5}

    ||文字列を連結する
    {sb.concat ", here comes..."}

    ||文字列を連結する
    let i:int = {sb.write-one-string " Curl!"}

    ||実行結果を確認してください
    {value sb}
}
```

実行結果

```
Hello computer world, here comes... Curl!
```

【文字の変更】1

サンプルコード

```
{value
    ||変数を定義する
    let sb1:StringBuf = {StringBuf "Here comes Curl!"}
    let sb2:StringBuf = {StringBuf}

    ||設定
    set sb2 = sb1

    ||文字を置き換える
    {sb2.set 5, 'C'}

    ||実行結果を確認してください
    {VBox sb1, sb2}
}
```

実行結果

```
Here Comes Curl!
Here Comes Curl!
```

【文字の変更】2

サンプルコード

```
{value
    ||変数を定義する
    let sb1:StringBuf = {StringBuf "Here comes Curl!"}
    let sb2:StringBuf = {StringBuf}

    ||内容を別の文字列の内容と置き換える
    {sb2.set-contents sb1}

    ||文字を置き換える
    {sb2.set 5, 'C'}

    ||実行結果を確認してください
    {VBox sb1, sb2}
}
```

実行結果

```
Here comes Curl!
Here Comes Curl!
```

【文字の変更】3

サンプルコード

```
{value
    ||変数を定義する
    let sb:StringBuf = {StringBuf "Hello World"}

    ||内容を別の文字列の内容と置き換える
    {sb.set-contents {StringBuf "Here comes Curl!"}}

     ||文字を置き換える
    {sb.set 5, 'C'}

    ||実行結果を確認してください
    {value sb}
}
```

実行結果

```
Here Comes Curl!
```

【文字の変更】4

サンプルコード

```
{value
    ||変数を定義する
    let sb:StringBuf = {StringBuf "Hello World"}

    ||配列構文を使用して、読み取りと書き込みが可能な文字列で個々の文字を変更する
    set sb[6] = 'w'

    ||実行結果を確認してください
    {value sb}
}
```

実行結果

```
Hello world
```

【文字の削除】

サンプルコード

```
||変数を定義する
{let sb:StringBuf = {StringBuf "Hello World"}}

||指定した文字を削除する
{sb.remove 2, length=7}

||実行結果を確認してください
{value sb}

||すべての文字を削除する
{sb.clear}

||実行結果を確認してください
{value sb}
```

実行結果

```
Held
```

【文字のトリミング】

サンプルコード

```
||変数を定義する
{let sb:StringBuf = {StringBuf "  -=-Hello World!!!  "}}

||実行結果を確認してください
{value sb}
|| Note that the display of whitespace is collapsed.

||文字列の先頭と末尾から文字を削除する
{sb.trim}

||実行結果を確認してください
{value sb}

||文字列の先頭から文字を削除する
{sb.trim-left trim-chars="-="}

||実行結果を確認してください
{value sb}

||文字列の末尾から文字を削除する
{sb.trim-right trim-chars="!"}

||実行結果を確認してください
{value sb}
```

実行結果

```
-=-Hello World!!!
-=-Hello World!!!
Hello World!!!
Hello World
```

【文字列の反転】

サンプルコード

```
{value
    ||変数を定義する
    let sb:StringBuf = {StringBuf "Hello World!"}

    ||文字列の文字を反転する
    {sb.reverse}

    ||実行結果を確認してください
    {value sb}
}
```

実行結果

```
!dlroW olleH
```

【文字列での作業(変換・分割・比較・表示・編集)】

【文字列の大文字または小文字の変更】

サンプルコード

```
||変数を定義する
{let sb:StringBuf = {StringBuf "Hello World!"}}

||文字列内の文字を小文字に変換する
{sb.to-lower}

||実行結果を確認してください
{value sb}

||文字列内の文字を大文字に変換する
{sb.to-upper}

||実行結果を確認してください
{value sb}
```

実行結果

```
hello world!
HELLO WORLD!
```

【double への変換】

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "365e-10"

    ||文字列から double 返す
    let number:double = {s.to-double}

    ||実行結果を確認してください
    number
}
```

実行結果

```
3.65e-008
```

【int への変換】

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "-0X16D"

    ||文字列から int を返す
    let number:int = {s.to-int}

    ||実行結果を確認してください
    number
}
```

実行結果

```
-365
```

【文字列 への変換】

サンプルコード

```
{value
    ||変数を定義する
    let sb:StringBuf = {StringBuf "Hello World!"}

    ||String に文字列を変換する
    let s:String = {sb.to-String}

    ||実行結果を確認してください
    {value s}
}
```

実行結果

```
Hello World!
```

【ストリーム への変換】

サンプルコード

```
||変数を定義する
{let s:String = "Hello World"}

||入力ストリームに文字列を変換する
||TextInputStreamの詳細はCurl開発者ガイドを確認してください
{let tis:TextInputStream = {s.to-InputStream}}

||実行結果を確認してください
{tis.read-one-string}

||クローズ処理を行う
{tis.close}
```

実行結果

```
Hello World
```

【文字列の分割】1

サンプルコード

```
{value
    ||変数を定義する
    let s:String = ""
    let a:StringArray = {s.split}

    ||実行結果を確認してください
    {value a.size}
}
```

実行結果

```
0
```

【文字列の分割】2

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "www.curl.com"
    let out:VBox = {VBox}

    ||指定した文字で文字列を分割する
    let a:StringArray = {s.split split-chars="."}


    ||分割した結果を変数に格納する
    {for i:int = 0 to (a.size - 1) do
        {out.add {value a[i]}}
    }

    ||実行結果を確認してください
    {value out}
}
```

実行結果

```
www
curl
com
```

【文字列の分割】3

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "www.curl.com"
    let out:VBox = {VBox}

    ||指定した文字で文字列を分割する
    let a:StringArray = {s.split split-chars=":"}

    ||分割した結果を変数に格納する
    {for i:int = 0 to (a.size - 1) do
        {out.add {value a[i]}}
    }

    ||実行結果を確認してください
    {value out}
}
```

実行結果

www.curl.com

【辞書比較】

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World!"
    let s2:String = "hello world!"

    {switch {s1.compare s2}  ||  2 つの文字列を比較する
     case -1 do
        {text String 1 is less than String 2.}
     case 0 do
        {text String 1 is equal to String 2.}
     else
        {text String 1 is greater than String 2.}
    }
}
```

実行結果

```
String 1 is less than String 2.
```

【文字列比較】

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World!"
    let s2:String = "hello world!"

    ||String が対象文字列以下かどうかを調べる
    {if {String-leq? s1, s2} then
        {text String 1 is less than or equal to String 2.}
     else
        {text String 1 is greater than String 2.}
    }
}
```

実行結果

```
String 1 is less than or equal to String 2.
```

【同等性のテスト】1

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World!"
    let s2:String = "Hello World!"

    ||2 つの文字列の内容が等しいかどうかを調べる
    {if {s1.equal? s2} then
        {text The contents of the strings are equal.}
     else
        {text The contents of the strings are NOT equal.}
    }
}
```

実行結果

```
The contents of the strings are equal.
```

【同等性のテスト】2

サンプルコード

```
{value
    ||変数を定義する
    let s1:String = "Hello World!"
    let s2:String = "Hello World!"

    ||2 つの文字列の内容が等しいかどうかを調べる
    {if s1 == s2 then
        {text The contents of the strings are equal.}
     else
        {text The contents of the strings are NOT equal.}
    }
}
```

実行結果

```
The contents of the strings are equal.
```

【文字列の表示】1

サンプルコード

```
{let i:int = 1234}
{let j:int = -1234}
{let k:int = 56}

||整数のフォーマットサンプルです
||詳細はCurl開発者ガイドを確認してください
{Table
    columns = 4,
    cell-border-width = 2pt,

    {text Unsigned Decimal Integer},
    {format "%u", i},
    {text N/A},
    {format "%u", k},

    {text Unsigned Octal Integer},
    {format "%o", i},
    {text N/A},
    {format "%o", k},

    {text ... with {monospace #} modifier},
    {format "%#o", i},
    {text N/A},
    {format "%#o", k},

    {text Unsigned Hexadecimal Integer},
    {format "%x", i},
    {text N/A},
    {format "%x", k},

    {text ... with {monospace #} modifier},
    {format "%#x", i},
    {text N/A},
    {format "%#x", k},

    {text Unsigned Binary Integer},
    {format "%b", i},
    {text N/A},
    {format "%b", k},

    {text ... with {monospace #} modifier},
    {format "%#b", i},
    {text N/A},
    {format "%#b", k}
}
```

実行結果

Unsigned Decimal Integer	1234	N/A	56
Unsigned Octal Integer	2322	N/A	70
... with # modifier	02322	N/A	070
Unsigned Hexadecimal Integer	4d2	N/A	38
... with # modifier	0x4d2	N/A	0x38
Unsigned Binary Integer	10011010010	N/A	111000
... with # modifier	0b10011010010	N/A	0b111000

【文字列の表示】2

サンプルコード

```
{let i:int = 1234}
{let j:int = -1234}
{let k:int = 56}

||整数のフォーマットサンプルです
||詳細はCurl開発者ガイドを確認してください
{Table
    columns = 4,
    cell-border-width = 2pt,
    {text Signed Decimal Integer},
    {format "%d", i},
    {format "%d", j},
    {format "%d", k},

    {text ... with {monospace +} modifier (sign)},
    {format "%+d", i},
    {format "%+d", j},
    {format "%+d", k},

    {text ... with {italic width} set to 1},
    {format "%1d", i},
    {format "%1d", j},
    {format "%1d", k},

    {text ... with {italic width} set to 8},
    {format "%8d", i},
    {format "%8d", j},
    {format "%8d", k},

    {text ... and {monospace 0} modifier (leading zeros)},
    {format "%08d", i},
    {format "%08d", j},
    {format "%08d", k}
}
```

実行結果

Signed Decimal Integer	1234	-1234	56
... with + modifier (sign)	+1234	-1234	+56
... with *width* set to 1	1234	-1234	56
... with *width* set to 8	1234	-1234	56
... and 0 modifier (leading zeros)	00001234	-0001234	00000056

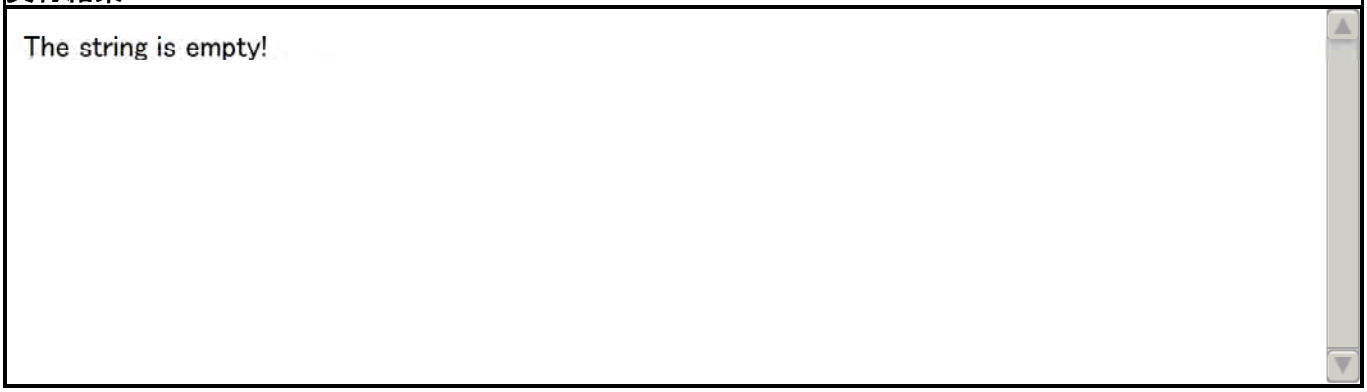
【空の文字列のテスト】

サンプルコード

```
{value
    ||変数を定義する
    let s:String = ""

    ||文字列が空かどうかを示す
    {if s.empty? then
        {text The string is empty!}
     else
        {text The string has characters!}
    }
}
```

実行結果



【文字列サイズの取得】

サンプルコード

```
{value
    ||変数を定義する
    let s:String = "Hello World!"

    ||文字列のサイズを示す
    {text There are {value s.size} characters in the string.}
}
```

実行結果

【文字列サイズの設定】

サンプルコード

```
||変数を定義する
{let sb:StringBuf = {StringBuf "Hello World!"}}

||文字列のサイズを設定する
{set sb.size = 5}

||実行結果を確認してください
{text If size is set to 5, the string
    becomes: {value sb}}

||文字列のサイズを設定する
{set sb.size = 10}

||実行結果を確認してください
{text Then, if size is set to 10, the string
    becomes: {value sb}}
```

実行結果

If size is set to 5, the string becomes: Hello

Then, if size is set to 10, the string becomes: Hello

【文字列の先頭を調べる】

サンプルコード

```
||変数を定義する
{let s:String = "Hello World!"}

||指定した文字で文字列が開始するかどうかを調べる
{value
    {s.prefix? "Hello"}
}
{value
    {s.prefix? "hello"}
}
{value
    {s.prefix? "hello", ignore-case?=true}
}
```

実行結果

```
true
false
true
```

【文字列の末尾を調べる】

サンプルコード

```
||変数を定義する
{let s:String = "Hello World!"}

||指定した文字で文字列が終了するかどうかを調べる
{value
    {s.suffix? "World!"}
}
{value
    {s.suffix? "world!"}
}
{value
    {s.suffix? "world!", ignore-case?=true}
}
```

実行結果

```
true
false
true
```

【文字列クローンの操作】

【文字のトリミング】1

サンプルコード

```
||変数を定義する
{let s1:String = "  -=-Hello World!!!  "}

||実行結果を確認してください
{value s1}

||変数を定義する
||クローンを作成して、文字列の最初と最後から文字を削除する
{let s2:String = {s1.trim-clone}}

||実行結果を確認してください
{value s2}

||変数を定義する
||クローンを作成して、文字列の最初から文字を削除する
{let s3:String = {s2.trim-left-clone trim-chars="-="}}

||実行結果を確認してください
{value s3}

||変数を定義する
||クローンを作成して、文字列の最後から文字を削除する
{let s4:String = {s3.trim-right-clone trim-chars="!"}}

||実行結果を確認してください
{value s4}
```

実行結果

```
-=-Hello World!!!
-=-Hello World!!!
Hello World!!!
Hello World
```

【文字のトリミング】2

サンプルコード

```
||変数を定義する
{let s1:String = "Hello World!"}

||変数を定義する
||クローンを作成して、含まれている文字を小文字に変換する
{let s2:String = {s1.to-lower-clone}}

||実行結果を確認してください
{value s2}

||変数を定義する
||クローンを作成して、含まれている文字を大文字に変換する
{let s3:String = {s2.to-upper-clone}}

||実行結果を確認してください
{value s3}
```

実行結果

```
hello world!
HELLO WORLD!
```