◆sortメソッド(構文：array.sort)

sortメソッドは、配列の要素を昇順にソートした新しい配列を返す。要素の順序の比較には<=>演算子が使用され、「要素1 <=> 要素2」の結果が-1なら要素1が先、0なら同じ、1なら要素2が先となる。ソートした結果を新しい配列で返すため、非破壊的メソッドと呼ばれる。

※ <=>で比較できない要素が混じっていると例外ArgumentErrorが、
要素の中に<=>を実装していないオブジェクトがあるとNoMethodErrorが発生する。

(例1：プログラム)

```
num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
print "元の配列：" + num.to_s + "\n"
print "ソート結果：" + (num.sort).to_s + “\n”
print "ソート後の配列の中身：" + num.to_s + "\n"
```

(例1：実行結果)

```
元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
ソート結果：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
ソート後の配列の中身：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
```

★ソートした結果は新しい配列で返されるため、元々宣言した時の配列の要素は変わらない

◆sort.reverseメソッド(構文：array.sort.reverse)

sort.reverseメソッドは、配列の要素を降順にソートした新しい配列を返す。その他の機能はsortメソッドと同じ。

(例2：プログラム)

```
num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
print "元の配列：" + num.to_s + "\n"
print "ソート結果：" + (num.sort.reverse).to_s + "\n"
print "ソート後の配列の中身：" + num.to_s + "\n"
```

(例2：実行結果)

```
元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
ソート結果：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
ソート後の配列の中身：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
```

★ソートした結果は新しい配列で返されるため、元々宣言した時の配列の要素は変わらない

◆ブロックを渡したsortメソッド(構文：array.sort { |a, b| block })

sortメソッドにブロックを渡すと、<=>演算子の代わりにブロックの戻り値によって要素をソートする。ブロック引数 a, b には、比較する2要素が入る。ブロックの戻り値が-1ならaが先、0なら同じ、1ならbが先となる。|a, b|はaが小さい値、bが大きい値の意味みたいな感じ。

(例3：プログラム)

```
num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
print "元の配列：" + num.to_s + "\n"
print "ソート結果：" + (num.sort { |a, b| a <=> b }).to_s + “\n"
print "ソート後の配列の中身：" + num.to_s + “\n"
```

(例3：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
　ソート後の配列の中身：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　★ソートした結果は新しい配列で返されるため、元々宣言した時の配列の要素は変わらない

(例4：プログラム)
```
　num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　print "元の配列：" + num.to_s + "\n"
　print "ソート結果：" + (num.sort { |a, b| b <=> a }).to_s + “\n"
　print "ソート後の配列の中身：" + num.to_s + “\n"
```
(例4：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
　ソート後の配列の中身：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　★ソートした結果は新しい配列で返されるため、元々宣言した時の配列の要素は変わらない

◆sort!メソッド(構文：array.sort!)
sort!メソッドは、配列の要素を昇順にソートする。レシーバ(配列)自身を変更するメソッドで、戻り値はレシーバ自身。sortメソッドと同じく順序の比較には<=>演算子が使われる。配列そのものをソートして変更するため、破壊的メソッドと呼ばれる。
(例5：プログラム)
```
　num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　print "元の配列：" + num.to_s + "\n"
　print "ソート結果：" + (num.sort!).to_s + "\n"
　print "ソート後の配列の中身：" + num.to_s + "\n"
```
(例5：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
　ソート後の配列の中身：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
　★配列自身をソートした結果を返すため、元々宣言した時の配列の要素が変更される

◆sort!.reverse!メソッド(構文：array.sort!.reverse!)
sort!メソッドは、配列の要素を降順にソートする。その他の機能はsort!メソッドと同じ。
(例6：プログラム)
```
　num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　print "元の配列：" + num.to_s + "\n"
　print "ソート結果：" + (num.sort!.reverse!).to_s + "\n"
　print "ソート後の配列の中身：" + num.to_s + “\n"
```

(例6：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
　ソート後の配列の中身：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
　★配列自身をソートした結果を返すため、元々宣言した時の配列の要素が変更される

◆ブロックを渡したsort!メソッド(構文：array.sort! { |a, b| block })
sort!メソッドにブロックを渡すと、ブロックを渡したsortメソッドと同じように処理をする。
(例7：プログラム)

```
num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
print "元の配列：" + num.to_s + "\n"
print "ソート結果：" + (num.sort! { |a, b| a <=> b }).to_s + “\n"
print "ソート後の配列の中身：" + num.to_s + “\n"
```

(例7：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
　ソート後の配列の中身：[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]
　★配列自身をソートした結果を返すため、元々宣言した時の配列の要素が変更される

(例8：プログラム)

```
num = [2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
print "元の配列：" + num.to_s + "\n"
print "ソート結果：" + (num.sort! { |a, b| b <=> a }).to_s + “\n"
print "ソート後の配列の中身：" + num.to_s + “\n"
```

(例8：実行結果)
　元の配列：[2, 3, 1, 4, 5, 1, 8, 3, 7, 6, 10, 9]
　ソート結果：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
　ソート後の配列の中身：[10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1]
　★配列自身をソートした結果を返すため、元々宣言した時の配列の要素が変更される