OSTR!

Er, SORT!

Alpha to Omega (or is it Omega to Alpha?)

What?

In this short talk, we'll look at sequences and how to make them orderly.

Sequences

- Arrays
- Dictionaries
- Sets
- Et cetera

Comparable

- **Sequences can be sorted via the `Comparable` protocol.**
- **This always returns an Array, regardless of initial collection type.**

But how do you pronounce it?!

Simple Mutating Sort

```
var otherStringedInstruments = ["Lanikai", "Kentucky", "Deering", "Woodrow"]
otherStringedInstruments.sort()
```

Simple Mutating Sort

```
var otherStringedInstruments = ["Lanikai", "Kentucky", "Deering", "Woodrow"]
otherStringedInstruments.sort()
```

["Deering", "Kentucky", "Lanikai", "Woodrow"]

Alternative Syntax

```
otherStringedInstruments.sort { (a, b) -> Bool in
    a < b
}
```

Alternative Syntax

```
otherStringedInstruments.sort {
    $0 < $1
}
```

Different Ordering

```
var guitars = [
    "Charvel", "Ovation", "Peavey",
    "Gibson", "Jackson", "Reverend",
    "Fender", "D'Angelico", "Schecter",
    "Ernie Ball"]
```

Different Ordering

```
guitars.sort()
guitars.reverse()
```

["Charvel", "D\'Angelico", "Ernie Ball", "Fender", "Gibson", "Jackson", "Ovation", "Peavey", "Reverend", "Schecter"]
["Schecter", "Reverend", "Peavey", "Ovation", "Jackson", "Gibson", "Fender", "Ernie Ball", "D\'Angelico", "Charvel"]

Alternate Syntax

```
guitars.sort(by: >)
```

Mixed Case?

```
var mixedCaseInstruments = [
    "Lanikai", "lanikai", "Kentucky", "kentucky",
    "Deering", "deering", "Woodrow", "woodrow"]
mixedCaseInstruments.sort()
```

Mixed Case?

```
var mixedCaseInstruments = [
    "Lanikai", "lanikai", "Kentucky", "kentucky",
    "Deering", "deering", "Woodrow", "woodrow"]
mixedCaseInstruments.sort()
```

["Deering", "Kentucky", "Lanikai", "Woodrow", "deering", "kentucky", "lanikai", "woodrow"]

What about Heterogeneity?

```
var otherOtherStringed = ["Lanikai", "Kentucky", "Deering", "Woodrow", nil]
otherOtherStringed.sort()
```

What about Heterogeneity?

```
var otherOtherStringed = ["Lanikai", "Kentucky", "Deering", "Woodrow", nil]
otherOtherStringed.sort()
```

"Referencing instance method 'sort()' on 'MutableCollection' requires that 'String?' conform to 'Comparable'"

What about Heterogeneity?

```
var otherOtherStringed = [
    "Lanikai", "Kentucky", "Deering", "Woodrow",
    "", "   ", "\t\n\t\n", "🤓"]
otherOtherStringed.sort()
```

What about Heterogeneity?

```
var otherOtherStringed = [
    "Lanikai", "Kentucky", "Deering", "Woodrow",
    "", "  ", "\t\n\t\n", "🤓"]
otherOtherStringed.sort()
```

["", "\t\n\t\n", " ", "Deering", "Kentucky", "Lanikai", "Woodrow", "🤓"]

More Mixed Case?

```
var beyondLatinIso = ["👀", "™", "°C", "¶", "Þ", "æ", "Ӝ"]
beyondLatinIso.sort()
```

More Mixed Case?

```
var beyondLatinIso = ["👀", "™", "°C", "¶", "฿", "æ", "Ӝ"]
beyondLatinIso.sort()
```

["¶", "æ", "Ӝ", "฿", "°C", "™", "👀"]

Sort*ed* and More

```
let otherInstruments = [
    "keyboard", "theremin", "cajon", "bongo",
    "flute", "recorder", "harmonica", "iPad"]
otherInstruments.sort()
```

Sort*ed* and More

```
let otherInstruments = [
    "keyboard", "theremin", "cajon", "bongo",
    "flute", "recorder", "harmonica", "iPad"]
otherInstruments.sort()
```

Error! mutating vs `let`

Sort*ed* and More

```
_ = otherInstruments.sorted()
_ = otherInstruments.sorted {
    $0.count < $1.count
}
_ = otherInstruments.reversed()
```

Review of the Simple Stuff

```
var justNumbers = [9, 5, 8, 2, 1, 3, 42, 0, -4, 99, 22]
justNumbers.sort()
justNumbers.sort { (a, b) -> Bool in
    a < b
}
justNumbers.sort {
    $0 < $1
}
justNumbers.sort(by: <)

justNumbers.reverse()
justNumbers.sort(by: >)

_ = justNumbers.sorted()
_ = justNumbers.reversed()
```

Structured Sorting

```
struct StringedInstrument {
    let name: String
    let stringCount: Int
}
```

Structured Sorting

```
let musicMan = StringedInstrument(name: "Baritone", stringCount: 6)
let soloist = StringedInstrument(name: "Jackson", stringCount: 7)
let t120 = StringedInstrument(name: "Bass", stringCount: 4)
let artist = StringedInstrument(name: "Dulcimer", stringCount: 4)
let goodtime = StringedInstrument(name: "Banjo", stringCount: 5)

let stringers = [musicMan, soloist, t120, artist, goodtime]
```

Structured Sorting

```
let instrumentsSorted = stringers.sorted {
    $0.stringCount < $1.stringCount
}
```

Structured Sorting

```
let instrumentsSorted = stringers.sorted {
    $0.stringCount < $1.stringCount
}
```

Bass: 4
Dulcimer: 4
Banjo: 5
Baritone: 6
Jackson: 7

Structured Sorting

```
struct DatedPercussion {
    let when: Date
}
let earlier = DatedPercussion(when: Date().addingTimeInterval(-3600))
let now = DatedPercussion(when: Date())
let later = DatedPercussion(when: Date().addingTimeInterval(3600))

let chronoBeats = [now, later, earlier]
```

Structured Sorting

```
chronoBeats.sorted {
    $0.when < $1.when
}
```

Structured Sorting

```
chronoBeats.sorted {
     $0.when < $1.when
}
```

2021-01-12 23:22:03 +0000,
2021-01-13 00:22:03 +0000,
2021-01-12 22:22:03 +0000

Multi-member Sort

```
struct ArbitraryExample {
    let ordinal: Int
    let when: Date
    let name: String
}
```

Multi-member Sort

```
let dateA = Date()
let dateB = Date().advanced(by: 3600)

let first = ArbitraryExample(ordinal: 79, when: dateA, name: "First")
let second = ArbitraryExample(ordinal: 137, when: dateB, name: "Second")
let third = ArbitraryExample(ordinal: 44, when: dateA, name: "Third")
let fourth = ArbitraryExample(ordinal: 55, when: dateA, name: "Fourth")
let fifth = ArbitraryExample(ordinal: 137, when: dateB, name: "Fifth")
let sixth = ArbitraryExample(ordinal: 137, when: dateB, name: "AAA Sixth")

let arbiters = [first, second, third, fourth, fifth,sixth]
```

Multi-member Sort :: Step One

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    return a.when < b.when
}
```

Multi-member Sort :: Step One

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    return a.when < b.when
}
```

First 79 2021-01-12 23:25:06 +0000,
Third 44 2021-01-12 23:25:06 +0000,
Fourth 55 2021-01-12 23:25:06 +0000,
Second 137 2021-01-13 00:25:06 +0000,
Fifth 137 2021-01-13 00:25:06 +0000,
AAA Sixth 137 2021-01-13 00:25:06 +0000

Multi-member Sort :: Step Two

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    guard a.when == b.when else {
        return a.when < b.when
    }
    // Secondary sort by Int
    return a.ordinal < b.ordinal
}
```

Multi-member Sort :: Step Two

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    guard a.when == b.when else {
        return a.when < b.when
    }
    // Secondary sort by Int
    return a.ordinal < b.ordinal
}
```

Third 44 2021-01-12 23:27:24 +0000,
Fourth 55 2021-01-12 23:27:24 +0000,
First 79 2021-01-12 23:27:24 +0000,
Second 137 2021-01-13 00:27:24 +0000,
Fifth 137 2021-01-13 00:27:24 +0000,
AAA Sixth 137 2021-01-13 00:27:24 +0000

Multi-member Sort :: Step Three

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    guard a.when == b.when else {
        return a.when < b.when
    }
    // Secondary sort by Int
    guard a.ordinal == b.ordinal else {
        return a.ordinal < b.ordinal
    }
    // Tertiary sort by String
    return a.name < b.name
}
```

Multi-member Sort :: Step Three

```
_ = arbiters.sorted { (a, b) -> Bool in
    // Primary sort by Date
    guard a.when == b.when else {
        return a.when < b.when
    }
    // Secondary sort by Int
    guard a.ordinal == b.ordinal else {
        return a.ordinal < b.ordinal
    }
    // Tertiary sort by String
    return a.name < b.name
}
```

Third 44 2021-01-12 23:28:26 +0000,
Fourth 55 2021-01-12 23:28:26 +0000,
First 79 2021-01-12 23:28:26 +0000,
AAA Sixth 137 2021-01-13 00:28:26 +0000,
Fifth 137 2021-01-13 00:28:26 +0000,
Second 137 2021-01-13 00:28:26 +0000

Conforming to Comparable

```
class Synth {
    let type: String
    let other: String
    required init(type: String, other: String = "") {
        self.type = type
        self.other = other
    }
}
extension Synth: Comparable {
    static func < (lhs: Synth, rhs: Synth) -> Bool {
        lhs.type < rhs.type
    }
    static func == (lhs: Synth, rhs: Synth) -> Bool {
        return lhs.type == rhs.type && lhs.other == rhs.other
    }
}
```

Conforming to Comparable

Synthesis details: https://reverb.com/news/10-types-of-synthesis

```
let subtractive = Synth(type: "Subtractive")
let additive = Synth(type: "Additive")
let fm = Synth(type: "Frequency Modulation")
let duplicateFM = Synth(type: "Frequency Modulation")
let granular = Synth(type: "Granular")
let sampled = Synth(type: "Sample-based")
let wavetable = Synth(type: "Wavetable")
let vector = Synth(type: "Vector")
let spectral = Synth(type: "Spectral")
let physical = Synth(type: "Physical Modeling")
let westCoast = Synth(type: "West Coast")
```

Conforming to Comparable

```
let synthesizers = [
    subtractive, additive, fm,
    duplicateFM, granular, sampled,
    wavetable, vector, spectral,
    physical, westCoast]
let sortedSynths = synthesizers.sorted()
```

Conforming to Comparable

```
let sortedSynths = synthesizers.sorted()
```

Additive, Frequency Modulation, Frequency Modulation, Granular, Physical Modeling, Sample-based, Spectral, Subtractive, Vector, Wavetable, West Coast

More from Equatable and Comparable

```
sortedSynths.contains(fm)
sortedSynths.contains(Synth(type: "Frequency Modulation", other: ""))
sortedSynths.contains(Synth(type: "Frequency Modulation", other: "different"))
```

true
true
false

More from Equatable and Comparable

```
sortedSynths.min()?.type
sortedSynths.max()?.type
```

Additive
West Coast

Questions? Answers? Additions? Corrections?

(Were the slides in the best sequence?)

Kevin Munc

@muncman