import UIKit

//: # Properties

/\*:

### Classification:

\* Stored properties : provided only by classes and structures

\* Computed properties : provided by classes, structures, and enumerations

\*/

//: ### Stored properties store constant and variable values as part of an instance

//: ### Computed properties calculate (rather than store) a value

//: ## Stored Properties

//: ### a stored property is a constant or variable that is stored as part of an instance of a particular class or structure

struct FixedLengthRange {

var firstValue: Int

let length: Int

}

var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3)

// the range represents integer values 0, 1, and 2

rangeOfThreeItems.firstValue = 6

// the range now represents integer values 6, 7, and 8

//: length is initialized when the new range is created and cannot be changed thereafter

//: ### Stored Properties of Constant Structure Instances

let rangeOfFourItems = FixedLengthRange(firstValue: 0, length: 4)

// this range represents integer values 0, 1, 2, and 3

//rangeOfFourItems.firstValue = 6

// this will report an error, even though firstValue is a variable property

//: ### Structures are value types

//: ### When an instance of a value type is marked as a constant, so are all of its properties

//: ## Computed Properties

//: ### They provide a getter and an optional setter to retrieve and set other properties and values indirectly

struct Point {

var x = 0.0, y = 0.0

}

struct Size {

var width = 0.0, height = 0.0

}

struct Rect {

var origin = Point()

var size = Size()

var center: Point {

get {

let centerX = origin.x + (size.width / 2)

let centerY = origin.y + (size.height / 2)

return Point(x: centerX, y: centerY)

}

set(newCenter) {

origin.x = newCenter.x - (size.width / 2)

origin.y = newCenter.y - (size.height / 2)

}

}

}

var square = Rect(origin: Point(x: 0.0, y: 0.0),

size: Size(width: 10.0, height: 10.0))

let initialSquareCenter = square.center

square.center = Point(x: 15.0, y: 15.0)

print("square.origin is now at (\(square.origin.x), \(square.origin.y))")

// Prints "square.origin is now at (10.0, 10.0)"

/\*:

![square](square.png)

\*/

//: ### Shorthand Setter Declaration

//: \* default name of \*newValue\* is used if there is no name for the computed property's setter

struct AlternativeRect {

var origin = Point()

var size = Size()

var center: Point {

get {

let centerX = origin.x + (size.width / 2)

let centerY = origin.y + (size.height / 2)

return Point(x: centerX, y: centerY)

}

set {

origin.x = newValue.x - (size.width / 2)

origin.y = newValue.y - (size.height / 2)

}

}

}

//: ### Read-Only Computed Properties

//: ### A computed property with a getter but no setter is known as a read-only computed property

//: ### a read-only computed property can be declared by removing the get keyword and its braces

struct Cuboid {

var width = 0.0, height = 0.0, depth = 0.0

var volume: Double {

return width \* height \* depth

}

}

let fourByFiveByTwo = Cuboid(width: 4.0, height: 5.0, depth: 2.0)

print("the volume of fourByFiveByTwo is \(fourByFiveByTwo.volume)")

// Prints "the volume of fourByFiveByTwo is 40.0"

//: ## Property Observers

//: ### Property observers observe and respond to changes in a property’s value

//: ### They are called every time a property’s value is set, even if the new value is the same as the property’s current value

/\*:

### Observers:

\* willSet is called just before the value is stored.

\* didSet is called immediately after the new value is stored.

\*/

class StepCounter {

var totalSteps: Int = 0 {

willSet(newTotalSteps) {

print("About to set totalSteps to \(newTotalSteps)")

}

didSet {

if totalSteps > oldValue {

print("Added \(totalSteps - oldValue) steps")

}

}

}

}

let stepCounter = StepCounter()

stepCounter.totalSteps = 200

// About to set totalSteps to 200

// Added 200 steps

stepCounter.totalSteps = 360

// About to set totalSteps to 360

// Added 160 steps

stepCounter.totalSteps = 896

// About to set totalSteps to 896

// Added 536 steps

//: ## Global and Local Variables

//: ### Global variables are variables that are defined outside of any function, method, closure, or type context

//: ### Local variables are variables that are defined within a function, method, or closure context

/\*:

### Variables:

\* Stored variables

\* Computed variables

\*/

//: ## Type Properties

//: ### properties that belong to the type itself

//: ### There will only ever be one copy of these properties, no matter how many instances of that type you create

/\*:

### Type Property Syntax:

\* static keyword

\* class keyword

\*/

struct SomeStructure {

static var storedTypeProperty = "Some value."

static var computedTypeProperty: Int {

return 1

}

}

enum SomeEnumeration {

static var storedTypeProperty = "Some value."

static var computedTypeProperty: Int {

return 6

}

}

class SomeClass {

static var storedTypeProperty = "Some value."

static var computedTypeProperty: Int {

return 27

}

class var overrideableComputedTypeProperty: Int {

return 107

}

}

//: ### Querying and Setting Type Properties

//: \* type properties are queried and set on the \*type\*, not on an instance of that type

print(SomeStructure.storedTypeProperty)

// Prints "Some value."

SomeStructure.storedTypeProperty = "Another value."

print(SomeStructure.storedTypeProperty)

// Prints "Another value."

print(SomeEnumeration.computedTypeProperty)

// Prints "6"

print(SomeClass.computedTypeProperty)

// Prints "27"