CS342/MED253 Building for Digital Health
Lecture 6A: Introduction to HealthKit
Overview for today

- Overview of other clinical studies
- Demo of VascTrac (https://vasctrac.stanford.edu)
- Introduction to HealthKit

https://cs342.stanford.edu

cs342-aut1920.slack.com
Assignment #4: Content and UI

Work on the look and feel of your application.
Finalize the content of the consent document and surveys with team mentors.

Spend next week finalizing the content for your app’s consent and surveys. We're posting this assignment as a set of guidelines and tips for some of the things that you can work on. We recommend that you have some of your team work on finalizing content, while others lend an eye to improving the user experience of study participants.

Take the following action items:

User Interface Action Items

1. **Create custom table view cells**
   Use Lecture 4A as a blueprint and customize your cells.
   Add an image, or some color.
use the resources we share on [github](https://github.com) freely
ResearchKit and CareKit Reimagined

The ResearchKit and CareKit frameworks are the fastest route to delivering powerful native iOS apps for the research and patient care field. Learn about the newest audio and speech active-tasks, a completely redesigned user experience, and modular architecture designed to make it a breeze to create care plan apps for any use case.
Health and Medicine Apps Using ResearchKit and CareKit

Now, everybody can better manage their personal health and contribute to advances in medical research.

We developed ResearchKit and CareKit to empower you to take a more active role in your health, well-being, and the future of medicine. These apps, paired with the advanced technologies built into your iPhone, help you manage your medical conditions, symptoms, and medications, and collect the type of information that will ultimately lead to new breakthroughs in medicine.
Welcome to PPD ACT
Postpartum Depression: Action towards Causes and Treatments
A Genetic Research Study of Postpartum Depression

Data Collection
This study is about postpartum depression. We are collecting data on experiences following childbirth. You may be asked to take part in the study.

Learn more

PPD ACT
University of North Carolina at Chapel Hill

Not Enough Ratings

Age

12+

Join the PACT

Welcome to PPD ACT
Postpartum Depression: Action towards Causes and Treatments
A Genetic Research Study of Postpartum Depression

Data Collection
This study is about postpartum depression. We are collecting data on experiences following childbirth. You may be asked to take part in the study.

Learn more
Welcome to NYU Langone's Concussion Tracker

A Concussion Research Study

Read Consent Document
Great Consent Document

Join Study
Already Participating?

Six-Minute Walking Activity

You will be asked to do a six-minute walking task. The objective of this test is to walk, at a comfortable pace, as far as possible in six minutes. You will be permitted to slow down, to stop, and to rest as necessary. You may rest, sit, or lean during the activity and restart if you wish. If at any time you feel light-headed or unwell during the activity, stop and immediately seek medical attention.
Consent Walkthrough

The next few screens will explain the research study and help you decide whether or not you want to participate.

After the walkthrough, we’ll briefly test your understanding to make sure you know what is involved.

Get Started
Apple’s new ResearchKit allows you to participate in important medical research studies easily through your iPhone. Stanford Medi...
MyGeneRank
Unlock your Genetic Risk

2.3 ★★★★★
25 Ratings

Age
17+

Dashboard
Activities Completed
25%

Coronary Artery Disease Risk
High levels. Risk 3x higher than average.

Active Time (Minutes)
Most day: 9:03; Average: 8
Personal Best: 19

Healthy Diet
10 year Risk for Heart Attack

10% risk: 1% Risk
25% risk: 4% Risk
50% risk: 7% Risk
75% risk: 10% Risk

Active Lifestyle
An active lifestyle is defined as at least 150 minutes of moderate exercise or 75 minutes of vigorous exercise per week.

MS Mosaic
A Multiple Sclerosis Research Study

3.1 ★★★★★
7 Ratings

12+

Read Consent Document
Email Consent Document

Activities
- Daily Survey
- Set Up Medications
- Self-Efficacy Survey
- Relapse Concern Survey
- Walking (Gait) Activity
- Tapping Activity
- 9 Hole Peg Activity

Initial Survey
Average: 8.866 sec

Today, Wednesday, Jan 25
To start an activity, select from the list below

Many conditions affecting our health are caused by a combination of our environment, our behaviors, and our genes. While we can all make changes for a healthier lifestyle, research shows that lifestyle can impact health outcomes. This risk estimate is only valid for an age of 20 or below. At 20 years of age, the risk estimate is 10.2.

MS Mosaic is a Duke sponsored research study that seeks to better understand multiple sclerosis from daily experiences. Like you.
ResearchKit and CareKit Reimagined

The ResearchKit and CareKit frameworks are the fastest route to delivering powerful native iOS apps for the research and patient care field. Learn about the newest audio and speech active-tasks, a completely redesigned user experience, and modular architecture designed to make it a breeze to create care plan apps for any use case.
Overview for today

- Overview of other clinical studies
- Demo of VascTrac (https://vasctrac.stanford.edu)
- Introduction to HealthKit

https://cs342.stanford.edu

cs342-aut1920.slack.com
Peripheral Artery Disease (PAD)

~ 10 million in US

Earliest symptom = **Claudication**

muscle cramping with exercise

Can progress to...

- rest pain
- wounds/gangrene
- amputations
Diagnosis

- Ankle-Brachial Index (great screening tool)
- Duplex Ultrasound (diagnostic + can visualize lesions)
- Angiogram (diagnostic + intervention)
- Patient History

Bypass Surgery

Endovascular Stent

Scar Tissue
Gaps in Surveillance and Management

**Problem:**
- High intervention failure rates
- Non personalized follow up

**Can we do better?**
- Personalized surveillance?
- Earlier failure detection?
- Better assessment of activity?
- Better outcomes?
Everything else, is a combination of these metrics.
iPhone as a Monitoring Tool: Healthy Population
iPhone as a Monitoring Tool: PAD Population
Accuracy Study (2016-2018)

n=182 patients w/ PAD getting ABIs
In-clinic 6MWT (100 ft Hallway Track)
Step Counting - i.e. study design continued

1. VascTrac → ActiGraph

2. ActiGraph → Manual Counting

3. VascTrac → Manual Counting
Steps: ActiGraph vs iPhone

Key Takeaways:
- iPhone is not worse than ActiGraph (mean difference = 10)

Steps: Manual vs. ActiGraph

Key Takeaways:
- Underestimates steps (mean = 20) & Higher variability (std = 53)


Accuracy: 95%±10%
Steps: Manual vs. iPhone

Key Takeaways:
- iPhone consistently underestimates (mean difference of 30)
- Lower variability (std = 32)
- Same story for sub analyses
Key Takeways:
- Phone overestimates distance (mean diff = -320) / stride length
- Error percentages higher for distance
ABI has low correlation to step count

Key Takeway:
- VascTrac shows low correlation found between ABI and steps

Accuracy Study Takeaways

1) Step counting was as good as actigraph
2) Distance counting was not as accurate (6MWT uses distance!)
3) Increasing errors at slower cadence with walkers/canes
Reliability / Repeatability Study (2018- Dec 2019)

N = 110 patients w/ planned interventions
  ● PAD
  ● Cardiac Bypass
  ● Open Aortic Valve Replacement
  ● TAVR

Protocol:
  ● Provide iPhone / Apple Watch
  ● Enroll 2 wks prior to procedure -> follow for 6 months
  ● In clinic 6MWTs enrollment, 1, 3, and 6 months
  ● Weekly at home 6MWT via notification reminder
Reliability / Repeatability Study (2018- Dec 2019)

AIMS
1) Reliability / Repeatability of home 6MWT
2) Sensitivity / Specificity passive activity
3) Develop an Activity Index (activity as a vital sign)

Data Collected
1) Daily steps / distance / MSWS
2) Weekly home 6MWT
3) Quarterly QOL survey, PAD survey, medical and surgical survey
   * 6MWT collect raw accelerometer data
### Users

<table>
<thead>
<tr>
<th>EID</th>
<th>Last active 2 days ago</th>
<th>Last walktest 2 days ago</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226221</td>
<td></td>
<td></td>
<td>10/29/2019, 7:51:17 AM</td>
</tr>
<tr>
<td>3335968</td>
<td></td>
<td></td>
<td>10/29/2019, 7:49:12 AM</td>
</tr>
<tr>
<td>3331218</td>
<td></td>
<td></td>
<td>10/29/2019, 7:45:09 AM</td>
</tr>
<tr>
<td>1074</td>
<td></td>
<td></td>
<td>10/02/2019, 6:50:22 PM</td>
</tr>
<tr>
<td>1222</td>
<td></td>
<td></td>
<td>10/25/2019, 10:18:01 AM</td>
</tr>
<tr>
<td>1198</td>
<td></td>
<td></td>
<td>10/29/2019, 7:35:43 AM</td>
</tr>
</tbody>
</table>
# Scheduled Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Scheduled Date</th>
<th>Scheduled Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walktest</td>
<td>9/30/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>9/23/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>9/16/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>9/09/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>9/02/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>8/26/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Walktest</td>
<td>8/19/2019</td>
<td>8:00:00 AM</td>
<td>Push sent</td>
</tr>
<tr>
<td>Medical Survey</td>
<td>8/15/2019</td>
<td>11:54:54 AM</td>
<td>Push pending</td>
</tr>
</tbody>
</table>
## Dashboard

### Event History

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Scheduled Date</th>
<th>Scheduled Time</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical Survey</td>
<td>8/15/2019</td>
<td>11:54:54 AM</td>
<td></td>
</tr>
<tr>
<td>Surgery Survey</td>
<td>8/15/2019</td>
<td>11:54:54 AM</td>
<td></td>
</tr>
<tr>
<td>Walk Survey</td>
<td>8/15/2019</td>
<td>11:54:54 AM</td>
<td></td>
</tr>
<tr>
<td>Walktest</td>
<td>2/25/2019</td>
<td>12:00:00 PM</td>
<td></td>
</tr>
</tbody>
</table>
Raw Daily Activity Data
Raw Daily Activity Data
7 day moving average x 2

Enter an EID (111xxxx) to properly view the patient's MSWS and Total Steps over a 3 month timeframe.

1113909

Display Watch Graphs

MSWS and Total Daily Steps for EID:

- Total Steps 7x7
- MSWS 7x7
- MSWS 14x14
- Intervention

Month

Number of Steps

- Nov 2018
- Jan 2019
- Mar 2019
- May 2019
- Jul 2019
- Sep 2019

0

5k

10k

15k
Enter an EID (111xxxx) to properly view the patient’s MSWS and Total Steps over a 3 month timeframe.

Enter: 1112434

Display Watch Graphs

MSWS and Total Daily Steps for EID:

- **Total Steps 7x7**
- **MSWS 7x7**
- **MSWS 14x14**
- **Intervention**
- **Event**

Number of Steps

- 10k
- 8k
- 6k
- 4k
- 2k
- 0

Month

- May 2019
- Jun 2019
- Jul 2019
- Aug 2019
- Sep 2019
Enter an EID (111:xxxx) to properly view the patient's MSWS and Total Steps over a 3 month timeframe.

1111809

Display Watch Graphs

MSWS and Total Daily Steps for EID:

Patient Notes:
10-19-18 (Intervention): L endart
04-03-19 (Event): injury (carcinoma)
CABG/ AVR Improve

Enter an EID (111xxxx) to properly view the patient’s MSWS and Total Steps over a 3 month timeframe.

1112113

Display Watch Graphs

MSWS and Total Daily Steps for EID:

Patient Notes:
01-31-19 (Intervention): AVR
CABG/ AVR Decline

Enter an EID (111xxxx) to properly view the patient’s MSWS and Total Steps over a 3 month timeframe.

1111682

Display Watch Graphs

MSWS and Total Daily Steps for EID:

Patient Notes:
02-19-19 (Intervention): CABG
06-25-19 (Event): 7/9: lost phone ~ 2 wks ago
01-14-19 (Event): CATH
TAVR Improve

Enter an EID (111xxxx) to properly view the patient’s MSWS and Total Steps over a 3 month timeframe.

MSWS and Total Daily Steps for EID:

- Total Steps 7x7
- MSWS 7x7
- MSWS 14x14
- Intervention
- Event
- Event
TAVR Decline

Enter an EID (111xxxx) to properly view the patient's MSWS and Total Steps over a 3 month timeframe.

1114991

Display Watch Graphs

MSWS and Total Daily Steps for EID:

Patient Notes:
11-29-18 (Intervention): TAVR
11-08-18 (Event): CATH
03-04-19 (Event): ER: rt elbow bump
Findings

1) Can monitor daily activity in real-time

2) Can effectively schedule active tasks (6MWT) remotely and notify with notifications
   a) Can monitor completion of task in real-time
   b) Can **monitor effort** (power) of activity (in development)
   c) Can provide real-time effort feedback to patient (in development)

3) One-on-one troubleshooting key to success

*Last patient exits in December*
Home Based Exercise Therapy

1. Health Education
2. Exercise Assessment
3. Goal Setting

Schedule Exercise
Monitor
Compliance
Report feedback

Live Coach Dashboard (option)

Doctor’s Office Dashboard (option)

Dx w/ PAD Claudication in clinic

SVS Activity App

in clinic follow up

auto-goal setting
Overview for today

- Overview of other clinical studies
- Demo of VascTrac ([https://vasctrac.stanford.edu](https://vasctrac.stanford.edu))
- Introduction to HealthKit

[https://cs342.stanford.edu](https://cs342.stanford.edu)
[cs342-aut1920.slack.com](cs342-aut1920.slack.com)
HealthKit enables us to read health and activity data from the iPhone — this includes step count, heart rate, flights climbed, calories burned, health records, and other similar metrics.
HealthKit is a store of health and fitness data.
HealthKit is a store of health and fitness data.
HealthKit is a store of health and fitness data.

Apps
- e.g. sleep tracking

CoreMotion
- accelerometer;
- gyro; etc
HealthKit is a store of health and fitness data.

Ext. Wearables

Apps

e.g. sleep tracking

CoreMotion

accelerometer; gyro; etc
HealthKit is a store of health and fitness data.

Ext. Wearables

Apps
- e.g. sleep tracking

CoreMotion
- accelerometer;
- gyro; etc

SSMART CARE-IT
- any iOS app

HealthKit is a store of health and fitness data.
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
accessing data in HealthKit
use one of three available methods

1. **Direct Access**
used for profile information that is rarely updated.
accessing data in HealthKit
use one of three available methods

1. Direct Access
2. Queries
   
everything else with entries over time is queried — from step count, to health records. Returns a snapshot.
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
   similar to a query, but with the option to receive a callback when more similar data becomes available.
Getting Authorization from HealthKit
let hkTypesToRead: Set<HKObjectType> = [
HKObjectType.quantityType(forIdentifier: .stepCount)!,
HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
HKObjectType.quantityType(forIdentifier: .heartRate)!,
HKObjectType.characteristicType(forIdentifier: .bloodType)!
]
HealthKit Data Types

HealthKit uses HKObjectType subclasses to identify the different types of data stored in HealthKit:

- **HKCharacteristicType** represents data that does not typically change over time (for example, blood type).
- **HKQuantityType** represents samples that contain a numeric value (for example calories consumed).
- **HKCategoryType** represents samples that contain an option from a short list of possible values (for example sleep analysis).
- **HKCorrelationType** represents complex samples that contain a number of quantity or category samples (for example, a food sample that includes a number of nutrition samples).
- **HKWorkoutType** represents a workout and its associated data (for more information, see Workouts).
- Other object types (for example, HKActivitySummaryType, HKDocumentType, and HKSeriesType) represent other, specialized data types.
HealthKit Data Types

HealthKit uses HKObjectType subclasses to identify the different types of data stored in HealthKit:

- **HKCharacteristicType** represents data that does not typically change over time (for example, blood type).

- **HKQuantityType** represents samples that contain a numeric value (for example, calories consumed).

- **HKCategoryType** represents samples that contain an option from a short list of possible values (for example, sleep analysis).

- **HKCorrelationType** represents complex samples that contain a number of quantity or category samples (for example, a food sample that includes a number of nutrition samples).

- **HKWorkoutType** represents a workout and its associated data (for more information, see Workouts).

- Other object types (for example, HKActivitySummaryType, HKDocumentType, and HKSeriesType) represent other, specialized data types.
let hkTypesToRead: Set<HKObjectType> = [
  HKObjectType.quantityType(forIdentifier: .stepCount)!,
  HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
  HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
  HKObjectType.quantityType(forIdentifier: .heartRate)!,
  HKObjectType.characteristicType(forIdentifier: .bloodType)!
]
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
    HKObjectType.quantityType(forIdentifier: .heartRate)!,
    HKObjectType.characteristicType(forIdentifier: .bloodType)!
]

var hkTypesToWrite: Set<HKSAMPLETYPE> = [
    HKobjectType.quantityType(forIdentifier: .stepCount)!,
    HKobjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKobjectType.quantityType(forIdentifier: .flightsClimbed)!,
    HKobjectType.quantityType(forIdentifier: .heartRate)!
]
HealthKit is a store of health and fitness data. Ext. Wearables

Apps
e.g. sleep tracking

CoreMotion
accelerometer; gyro; etc

SSMART CARE-IT
any iOS app

HealthKit is a store of health and fitness data.
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
    HKObjectType.quantityType(forIdentifier: .heartRate)!,
    HKObjectType.characteristicType(forIdentifier: .bloodType)!
]

var hkTypesToWrite: Set<HKSAMPLEType> = [
    HKObject.quantityType(forIdentifier: .stepCount)!,
    HKObject.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObject.quantityType(forIdentifier: .flightsClimbed)!,
    HKObject.quantityType(forIdentifier: .heartRate)!
]
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
    HKObjectType.quantityType(forIdentifier: .heartRate)!,
    HKObjectType.characteristicType(forIdentifier: .bloodType)!
]

var hkTypesToWrite: Set<HKSAMPLETYPE> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!,
    HKObjectType.quantityType(type(forIdentifier: .heartRate))!
]

func getHealthAuthorization(completion: @escaping (_ success: Bool, _ error: Error?) -> Void) {
    guard HKHealthStore.isHealthDataAvailable() else {
        completion(false, HealthKitError.notAvailable)
        return
    }
    let healthStore = HKHealthStore()
    healthStore.requestAuthorization(toShare: hkTypesToWrite, read: hkTypesToRead) {
        (success, error) in
        completion(success, error)
    }
}
Getting Authorization from HealthKit
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.characteristicType(forIdentifier: .bloodType)!,
    HKObjectType.characteristicType(forIdentifier: .biologicalSex)!
]
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.characteristicType(forIdentifier: .bloodType)!,
    HKObjectType.characteristicType(forIdentifier: .biologicalSex)!
]

extension HKCharacteristicTypeIdentifier {

    @available(iOS 8.0, *)
    public static let biologicalSex: HKCharacteristicTypeIdentifier

    @available(iOS 8.0, *)
    public static let bloodType: HKCharacteristicTypeIdentifier // HKBloodTypeObject

    @available(iOS 8.0, *)
    public static let dateOfBirth: HKCharacteristicTypeIdentifier // NSDateComponents

    @available(iOS 9.0, *)
    public static let fitzpatrickSkinType: HKCharacteristicTypeIdentifier // HKFitzpatrickSkinTypeObject

    @available(iOS 10.0, *)
    public static let wheelchairUse: HKCharacteristicTypeIdentifier // HKWheelchairUseObject
}
let hkTypesToRead: Set<HKObjectType> = [
  HKObjectType.characteristicType(forIdentifier: .bloodType)!,
  HKObjectType.characteristicType(forIdentifier: .biologicalSex)!
]
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.characteristicType(forIdentifier: .bloodType)!,
    HKObjectType.characteristicType(forIdentifier: .biologicalSex)!
]

func getBloodType() throws -> HKBloodType {
    let healthStore = HKHealthStore()
    return try healthStore.bloodType().bloodType
}

func getBiologicalSex() throws -> HKBiologicalSex {
    let healthStore = HKHealthStore()
    return try healthStore.biologicalSex().biologicalSex
}
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
Sample Query
General-purpose. You will be using these most!

Anchored Query
Only return new/modified elements.

Statistics Query
Quickly find the sum, min, max, or avg of a data set.

Activity Query
Move, exercise, and stand data. Three rings.

Document Query
Useful to find health records!

More Information
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!
]
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount),!
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning),!
    HKObjectType.quantityType(forIdentifier: .flightsClimbed),!
]

func query(quantityType: HKQuantityTypeIdentifier, completion: @escaping ([HKQuantitySample], Error?) -> Void) {
    let predicate = HKQuery.predicateForSamples(withStart: Date(), yesterday, end: Date(), options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleType = HKObjectType.quantityType(forIdentifier: quantityType)!

    let query = HKSampleQuery(sampleType: sampleType, predicate: predicate, limit: 100, sortDescriptors: [sortDescriptor])
    (query, samples, error) in

    guard let samples = samples as? [HKQuantitySample] else {
        completion([HKQuantitySample](), error)
        return
    }

    completion(samples, error)
}

let healthStore = HKHealthStore()
healthStore.execute(query)
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!
]

func query(quantityType: HKQuantityTypeIdentifier, completion: @escaping ([HKQuantitySample], Error?) -> Void) {

    let predicate = HKQuery.predicateForSamples(withStart: Date(), yesterday, end: Date(), options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleType = HKObjectType.quantityType(forIdentifier: quantityType)!

    let query = HKSampleQuery(sampleType: sampleType, predicate: predicate, limit: 100, sortDescriptors: [sortDescriptor]) {
        (query, samples, error) in
            guard let samples = samples as? [HKQuantitySample] else {
                completion([HKQuantitySample](), error)
                return
            }
            completion(samples, error)
    }

    let healthStore = HKHealthStore()
    healthStore.execute(query)
Anchored Query
Only return new/modified elements.

Statistics Query
Quickly find the sum, min, max, or avg of a data set.

Activity Query
Move, exercise, and stand data. Three rings.

Document Query
Useful to find health records!

More Information
accessing data from HK using anchor queries
var myAnchor: HKQueryAnchor? = nil
func anchoredQueryStepCount(completion: @escaping ([HKQuantitySample], Error?) -> Void) {

    // Create the step count type.
    guard let stepCountType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        fatalError("*** Unable to get the step count type ***")
    }

    // Create the query.
    let query = HKAnchoredObjectQuery(type: stepCountType, predicate: nil, anchor: myAnchor, limit: HKObjectQueryNoLimit)
    { (query, samplesOrNil, deletedObjectsOrNil, newAnchor, errorOrNil) in
        guard let samples = samplesOrNil, let deletedObjects = deletedObjectsOrNil else {
            // Handle the error here.
            fatalError("*** An error occurred during the initial query: \(errorOrNil!.localizedDescription) ***")
        }

        self.myAnchor = newAnchor

        for stepCountSample in samples {
            // Process the new step count samples here.
        }

        for deletedStepCountSamples in deletedObjects {
            // Process the deleted step count samples here.
        }
    }

    // Optionally, add an update handler.
    query.updateHandler = {
        //...*
    }

    // Run the query.
    let healthStore = HKHealthStore()
    healthStore.execute(query)
var myAnchor:HKQueryAnchor? = nil
func anchoredQueryStepCount(completion: @escaping ([HKQuantitySample], Error?) -> Void) {
    // Create the step count type.
    guard let stepCountType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        // This should never fail when using a defined constant.
        fatalError("*** Unable to get the step count type ***")
    }
    // Create the query.
    let query = HKAnchoredObjectQuery(type: stepCountType, predicate: nil, anchor: myAnchor, limit: HKObjectQueryLimit)
    { (query, samplesOrNil, deletedObjectsOrNil, newAnchor, errorOrNil) in
        guard let samples = samplesOrNil, let deletedObjects = deletedObjectsOrNil else {
            // Handle the error here.
            fatalError("*** An error occurred during the initial query: \(errorOrNil!.localizedDescription) ***")
        }
        self.myAnchor = newAnchor
        for stepCountSample in samples {
            // Process the new step count samples here.
        }
        for deletedStepCountSamples in deletedObjects {
            // Process the deleted step count samples here.
        }
    }
    // Optionally, add an update handler.
    query.updateHandler = {
        //...*
    }
    // Run the query.
    let healthStore = HKHealthStore()
    healthStore.execute(query)
}
accessing data from HK
(2) using queries

Statistics Query
Quickly find the sum, min, max, or avg of a data set.

Activity Query
Move, exercise, and stand data. Three rings.

Document Query
Useful to find health records!

More Information
accessing data from HK using stats queries
func totalSteps(completion: @escaping (Double?, Error?) -> (Void)) {
    let calendar = NSCalendar.current
    let now = Date()
    let components = calendar.dateComponents([.day, .month, .year], from: now)
    let startDate = calendar.date(from: components)!
    let endDate = calendar.date(byAdding: .day, value: 1, to: startDate)
    let sampleType = HKQuantityType.quantityType(forIdentifier: .stepCount)!
    let predicate = HKQuery.predicateForSamples(withStart: startDate, end: endDate, options: .strictStartDate)
    let query = HKStatisticsQuery(
        quantityType: sampleType,
        quantitySamplePredicate: predicate,
        options: .cumulativeSum) {
            query, result, error in
                if error != nil {
                    completion(nil, error)
                    return
                }
                var total = 0.0
                if let quantity = result?.sumQuantity() {
                    let unit = HKUnit.count()  
                    total = quantity.doubleValue(for: unit)
                }
                completion(total, error)
    }
    let healthStore = HKHealthStore()
    healthStore.execute(query)
Accessing data from HK using queries

Activity Query
Move, exercise, and stand data. Three rings.

Document Query
Useful to find health records!

More Information
accessing data from HK using activity queries
func activityRings() {

    // Create the predicate for the query
    let summariesWithinRange = HKQuery.predicate(forActivitySummariesBetweenStart: startDate, end: endDate)

    // Build the query
    let query = HKActivitySummaryQuery(predicate: summariesWithinRange) { (query, summaries, error) -> Void in
        guard let activitySummaries = summaries else {
            guard let queryError = error else {
                fatalError("*** Did not return a valid error object. ***")
            }
        }

        // Handle the error here...
        return
    }

    // Do something with the summaries here...
    //activeEnergyBurned, appleExerciseTime, appleStandHours, activeEnergyBurnedGoal, appleExerciseTimeGoal, appleStandHoursGoal: HKQuantity

    // Run the query
    let healthStore = HKHealthStore()
    healthStore.execute(query)
```
accessing data from HK
(2) using queries

More Information

Document Query
Useful to find health records!
accessing data from HK health records
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.clinicalType(forIdentifier: .allergyRecord)!,
    HKObjectType.clinicalType(forIdentifier: .immunizationRecord)!
]
let hkTypesToRead: Set<HKObjectType> = [
HKObjectType.clinicalType(forIdentifier: .allergyRecord)!,
HKObjectType.clinicalType(forIdentifier: .immunizationRecord)!
]

extension HKClinicalTypeIdentifier {

@available(iOS 12.0, *)
public static let allergyRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let conditionRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let immunizationRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let labResultRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let medicationRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let procedureRecord: HKClinicalTypeIdentifier

@available(iOS 12.0, *)
public static let vitalSignRecord: HKClinicalTypeIdentifier
}
"HealthKit-Sample" would like to access your Health Records.

You should make sure that you trust this app before you grant access.

- The records you share may identify you and your care team.
- The app may be able to infer other aspects of your health history from the records you share.
- The app developer may retain and use your records after you stop sharing or delete the app.

**View Your Health Records**

**Continue**

**Not Now**

**Which categories can "HealthKit-Sample" access?**

**APP EXPLANATION**
Used for data security purposes.

**View "HealthKit-Sample" Privacy Policy**
Review the app’s Privacy Policy to understand how it may use and disclose your records.

**ALLOW "HEALTHKIT-SAMPLE" TO READ**
- Allergies
- Immunizations

"HealthKit-Sample" will have access to your current records as of October 28, 2019 at 7:10 PM and will automatically receive access to new records.

**SHARING NEW RECORDS**

- Ask Before Sharing
- Automatically Share

**How would you like to share new records?**

**Share Current Records**

- Don't Share

**Done**

**accessing data from HK health records 🏥**
```swift
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.clinicalType(forIdentifier: .allergyRecord)!,
    HKObjectType.clinicalType(forIdentifier: .immunizationRecord)!
]

func healthRecordAllergies() {
    guard let allergyType = HKObjectType.clinicalType(forIdentifier: .allergyRecord) else {
        fatalError("*** Unable to create the allergy type ***")
    }

    let allergyQuery = [HKSampleQuery](sampleType: allergyType, predicate: nil, limit: HKObjectQueryNoLimit, sortDescriptors: nil) { (query, samples, error) in
        guard let actualSamples = samples else {
            // Handle the error here.
            print("*** An error occurred: \(error?.localizedDescription ?? "nil") ***")
            return
        }

        let allergySamples = actualSamples as? [HKClinicalRecord]
        // Do something with the allergy samples here...
    }

    let store = HKHealthStore()
    store.execute(allergyQuery)
}
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
Observer Query
Notifies when samples change

Anchored Query
Only return new/modified elements.

Statistics Collection Query
Multiple statistic queries over time

Activity Query
Move, exercise, and stand data. Three rings.

More Information
Observer Query
Notifies when samples change

```swift
func observerQuery() {
    let sampleType = HKObjectType.quantityType(forIdentifier: .stepCount)!

    let query = HKObserverQuery(sampleType: sampleType, predicate: nil) {
        query, completionHandler, error in

        if error != nil {
            // Perform error handling.
            fatalError("unable to run query \(error!.localizedDescription)"")
        }

        // Take whatever steps are necessary to update your app's data and UI
        // This may involve executing other queries
        //self.updateDailyStepCount()

        // If you have subscribed for background updates you must call the completion handler here.
        completionHandler()
    }

    let store = HKHealthStore()
    store.execute(query)
}
```
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!
]

func query(quantityType: HKQuantityTypeIdentifier, completion: @escaping ([HKQuantitySample], Error?) -> Void) {
    let predicate = HKQuery.predicateForSamples(withStart: Date(), yesterday, end: Date(), options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleType = HKObjectType.quantityType(forIdentifier: quantityType)!

    let query = HKSampleQuery(sampleType: sampleType, predicate: predicate, limit: 100, sortDescriptors: [sortDescriptor])
    guard let samples = query.samples as? [HKQuantitySample] else {
        completion([HKQuantitySample](), error)
        return
    }

    completion(samples, error)
}

let healthStore = HKHealthStore()
healthStore.execute(query)
Anchored Query
Only return new/modified elements.

Statistics Collection Query
Multiple statistic queries over time

Activity Query
Move, exercise, and stand data. Three rings.

More Information
Accessing data from HK

(3) Long queries

Anchored Query
Only return new/modified elements.

Statistics Collection Query
Multiple statistic queries over time

Activity Query
Move, exercise, and stand data. Three rings.

More Information
var myAnchor: HKQueryAnchor? = nil
func anchoredQueryStepCount(completion: @escaping ([HKQuantitySample], Error?) -> Void) {
    // Create the step count type.
    guard let stepCountType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        // This should never fail when using a defined constant.
        fatalError("*** Unable to get the step count type ***")
    }

    // Create the query.
    let query = HKAnchoredObjectQuery(type: stepCountType, predicate: nil, anchor: myAnchor, limit: HKObjectQueryNoLimit)
    { (query, samplesOrNil, deletedObjectsOrNil, newAnchor, errorOrNil) in
        guard let samples = samplesOrNil, let deletedObjects = deletedObjectsOrNil else {
            // Handle the error here.
            fatalError("*** An error occurred during the initial query: \(errorOrNil!.localizedDescription) ***")
        }
        self.myAnchor = newAnchor
        for stepCountSample in samples {
            // Process the new step count samples here.
        }
        for deletedStepCountSamples in deletedObjects {
            // Process the deleted step count samples here.
        }
    }

    // Optionally, add an update handler.
    query.updateHandler = { /*...*/ }

    // Run the query.
    let healthStore = HKHealthStore()
    healthStore.execute(query)
accessing data from HK
(3) long queries

Statistics Collection Query
Multiple statistic queries over time

Activity Query
Move, exercise, and stand data. Three rings.

More Information
Statistics Collection Query
Multiple statistic queries over time

accessing data from HK
(3) long queries

Getting and Setting Results Handlers

```
var initialResultsHandler: ((HKStatisticsCollectionQuery, HKStatisticsCollection?, Error?) -> Void)?

    The results handler for this query’s initial results.
```

```
var statisticsUpdateHandler: ((HKStatisticsCollectionQuery, HKStatistics?, HKStatisticsCollection?, Error?) -> Void)?

    The results handler for monitoring updates to the HealthKit store.
```
Activity Query
Move, exercise, and stand data. Three rings.

Getting Property Data
var updateHandler: ((HKActivitySummaryQuery, [HKActivity Summary]?, Error?) -> Void)?

The handler for monitoring updates to activity summaries saved in the HealthKit store.
accessing data in HealthKit

use one of three available methods

1. Direct Access
2. Queries
3. Long-Running Queries
patient monitoring w/ HealthKit
we can respond to some changes in the health store live as they occur
Observer Query
Notifies when samples change

Anchored Query
Only return new/modified elements.

Statistics Collection Query
Multiple statistic queries over time

Activity Query
Move, exercise, and stand data. Three rings.

More Information
patient monitoring w/ HealthKit
we can respond to some changes in the health store live as they occur

Observer Queries can run 100% in the background
this means that in a clinical study, you can get meaningful data even if the patient rarely uses their phone.
activity detected
activity detected → phone wakes up
activity detected → phone wakes up → SSMART CARE-IT launches and (quietly) runs observer queries
Observer Query
Notifies when samples change

```swift
func observerQuery() {
    let sampleType = HKObjectType.quantityType(forIdentifier: .stepCount)!
    let query = HKObserverQuery(sampleType: sampleType, predicate: nil) {
        query, completionHandler, error in
            if error != nil {
                // Perform error handling.
                fatalError("unable to run query \(error!.localizedDescription)"
            }
            // Take whatever steps are necessary to update your app's data and UI
            // This may involve executing other queries
            //self.updateDailyStepCount()

            // If you have subscribed for background updates you must call the completion handler here.
            completionHandler()
    }
    let store = HKHealthStore()
    store.execute(query)
}
```
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount),
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning),
    HKObjectType.quantityType(forIdentifier: .flightsClimbed),
]

func query(quantityType: HKQuantityTypeIdentifier, completion: @escaping ([HKQuantitySample], Error?) -> Void) {
    let predicate = HKQuery.predicateForSamples(withStart: Date(), yesterday, end: Date(), options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleType = HKObjectType.quantityType(forIdentifier: quantityType)!

    let query = HKSampleQuery(sampleType: sampleType, predicate: predicate, limit: 100, sortDescriptors: [sortDescriptor])
    guard let samples = query.execute()!.samples as? [HKQuantitySample] else {
        completion([], nil)
        return
    }
    completion(samples, nil)
}

let healthStore = HKHealthStore()
healthStore.execute(query)
let hkTypesToRead: Set<HKObjectType> = [
    HKObjectType.quantityType(forIdentifier: .stepCount)!,
    HKObjectType.quantityType(forIdentifier: .distanceWalkingRunning)!,
    HKObjectType.quantityType(forIdentifier: .flightsClimbed)!
]

fileprivate func setUpBackgroundDelivery(forTypes types: Set<HKQuantityType>) {
    for type in types {
        let query = HKObserverQuery(sampleType: type, predicate: nil, updateHandler: { (query, completionHandler, error) in
            /*Handle Query & call completionHandler when done*/
        })

        healthStore.execute(query)
        healthStore.enableBackgroundDelivery(for: type, frequency: .immediate, withCompletion: { (success, error) in
            if let error = error {
                let error = error.localizedDescription
                print(error)
            }
        })
    }
}
activity detected

phone wakes up

launches and (quietly) runs observer queries

SSSMART CARE-IT
patient monitoring w/ HealthKit
we can respond to some changes in the health store live as they occur

You can use background time to:

1. Upload data for analysis
2. Show a notification to complete a survey
3. Detect that the user is going for a walk and suggest that they do so via an active task (6MWT) *
4. Many more things...
Attendance

- https://forms.gle/ygnFfzyMxgg4NKvC8

https://cs342.stanford.edu

cs342-aut1920.slack.com