CS123 - BLE & API

Programming Your Personal Robot

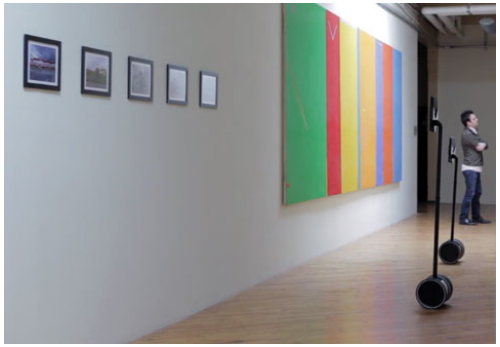
Kyong-Sok “KC” Chang, David Zhu
Fall 2015-16

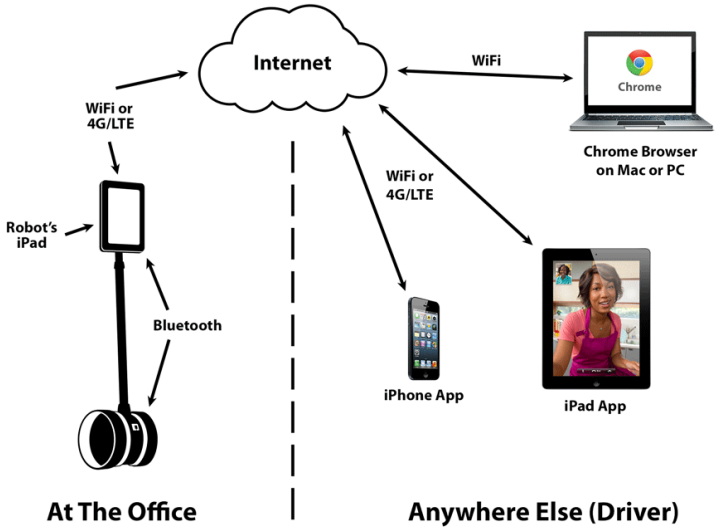
Logistics

- Getting your own Hamster (and USB BLE dongle)
  - Sign-up sheet
- Programming environment
  - Mac: Assignment#1-1 and #1-2 (must)
  - PC: only okay from Assignment#2 with USB BLE dongle
- Website for the class
  - cs123.stanford.edu, piazza
- TA sessions (office hours): this week
  - Location: Gates B21
  - Time: M:2~4pm, Tu:6~8pm, W:12:30-2:30pm, Th:6:30~8:30pm
- Lab reserved for cs123
  - MTuW, 12~6pm @ Gates B21

Future: Telepresence Robots?

From Left: AnyBots QB, RoboDynamics TiLR, Gostai Jazz Connect, Mantaro's Mantaro Bot, and VGo





Syllabus

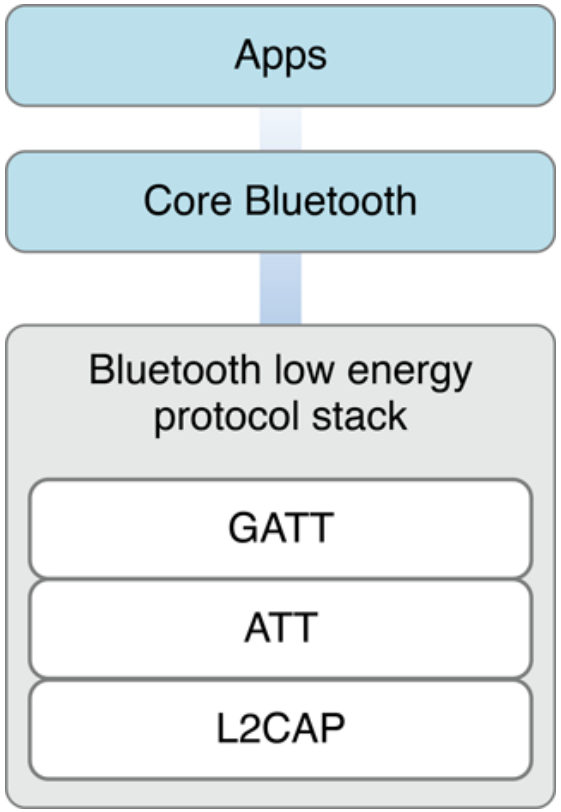
- Part 1 - Communicating with robot (2 weeks)
  - BLE communication and robot API
- Part 2 - Event Driven Behavior (2 weeks)
  - Finite State Machine (Behavior Tree)
- Part 3 - Reasoning with Uncertainty (2 weeks)
  - Dealing with noisy data, uncertainty in sensing and control
- Part 4 - Extending the robot (1 weeks)
  - I/O extensions: digital, analog, servo, pwm, etc
- Part 5 – Putting it together (including UI/UX) (3 weeks)
  - Design and implement of final (group) project
  - Encourage you to go “above and beyond”

Outline

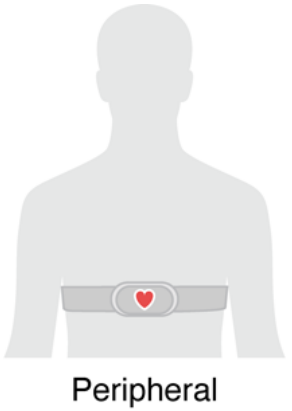
- BLE: Review - Core Bluetooth
- Delegation vs Protocol
- Assignment#1-1
- API: Definition
- Interface vs Implementation
- Naming Convention
- Robot API: Accessor vs Mutator
- Bytes and Bits
- Bitwise operators and masks
- Two’s complement
- Assignment#1-2

BLE: Core Bluetooth by Apple

Overview

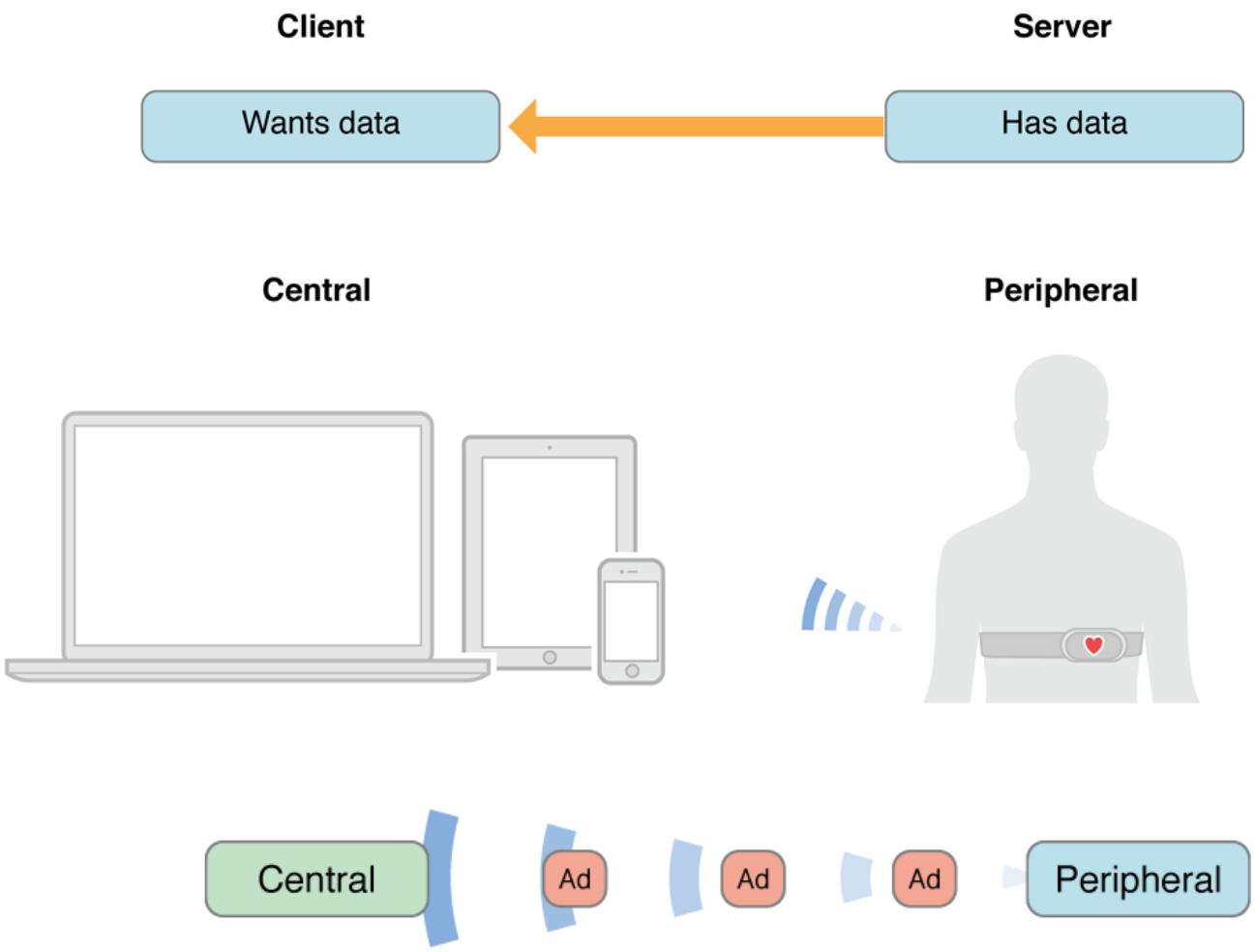


GATT Profile



Ref. Apple, Inc.

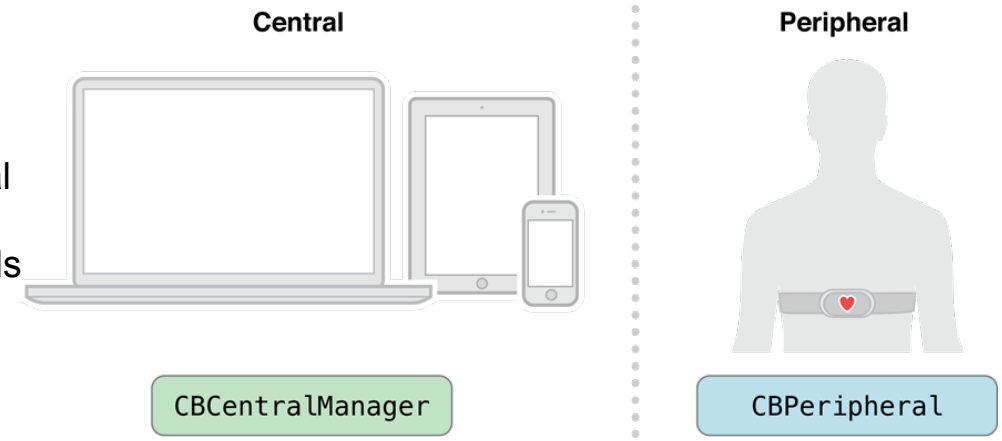
BLE: Roles



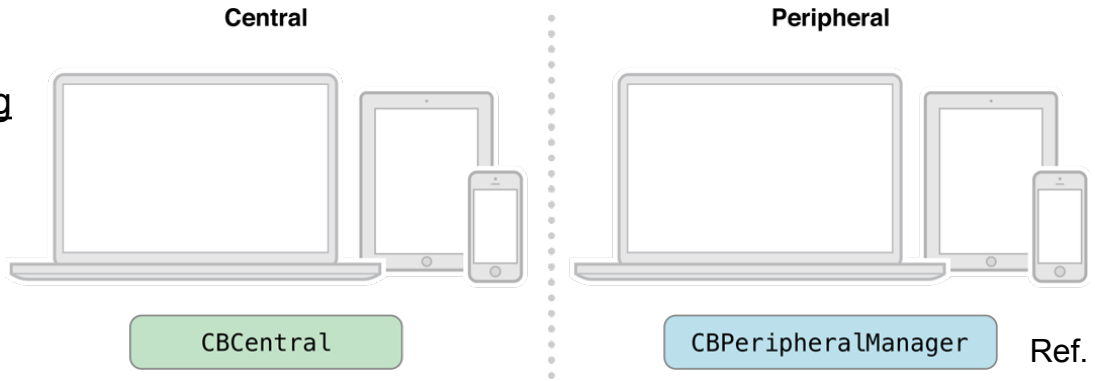
Ref. Apple, Inc.

Central vs Peripheral Programming

Central(host) programming
-Core Bluetooth objects on the central side
-Local centrals and remote peripherals

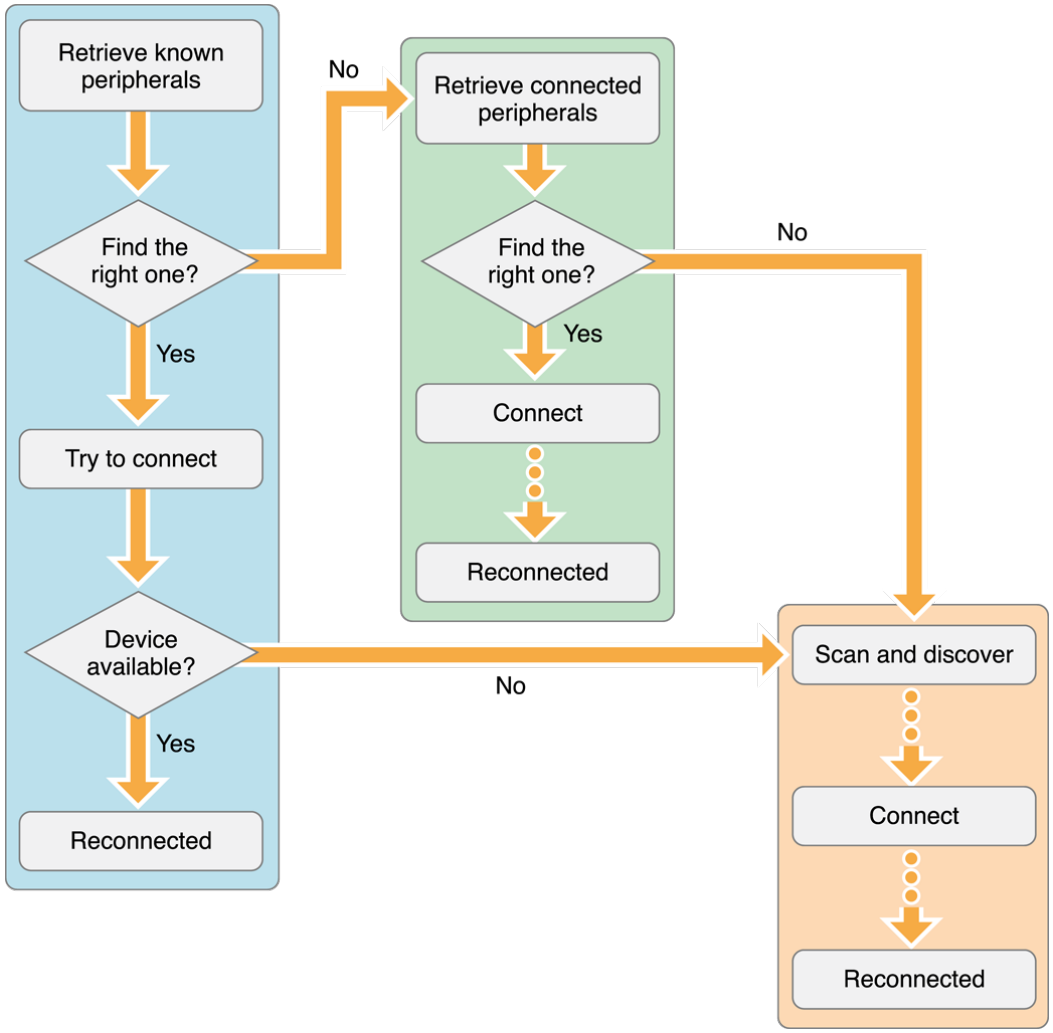


Peripheral(device) programming
-Core Bluetooth objects on the peripheral side
-Local peripherals and remote centrals



Ref. Apple, Inc.

Reconnection without scan

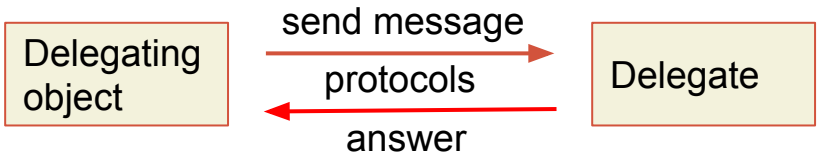


Ref. Apple, Inc.

Delegation vs Protocol

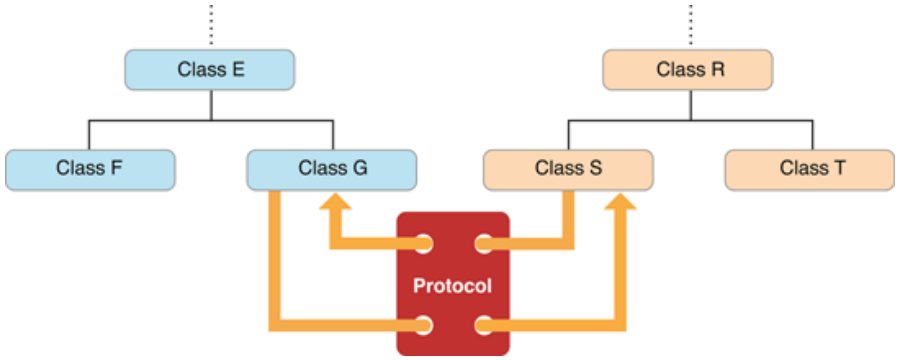
Delegation: Acting on Behalf of Another Object

In delegation, an object called the *delegate* acts on behalf of, and at the request of, another object. That other, delegating, object is typically a framework object. At some point in execution, it sends a message to its delegate; the message tells the delegate that some event is about to happen and asks for some response. The delegate (usually an instance of a custom class) implements the method invoked by the message and returns an appropriate value. Often that value is a Boolean value that tells the delegating object whether to proceed with an action. The delegating object has to keep track of the delegate and call upon it when needed by sending it a message.



Protocol: Enabling Communication Between Objects Not Related by Inheritance

A protocol is a declaration of a programmatic interface whose methods any class can implement. A protocol is thus, as is delegation, an alternative to subclassing and is often part of a framework's implementation of delegation.



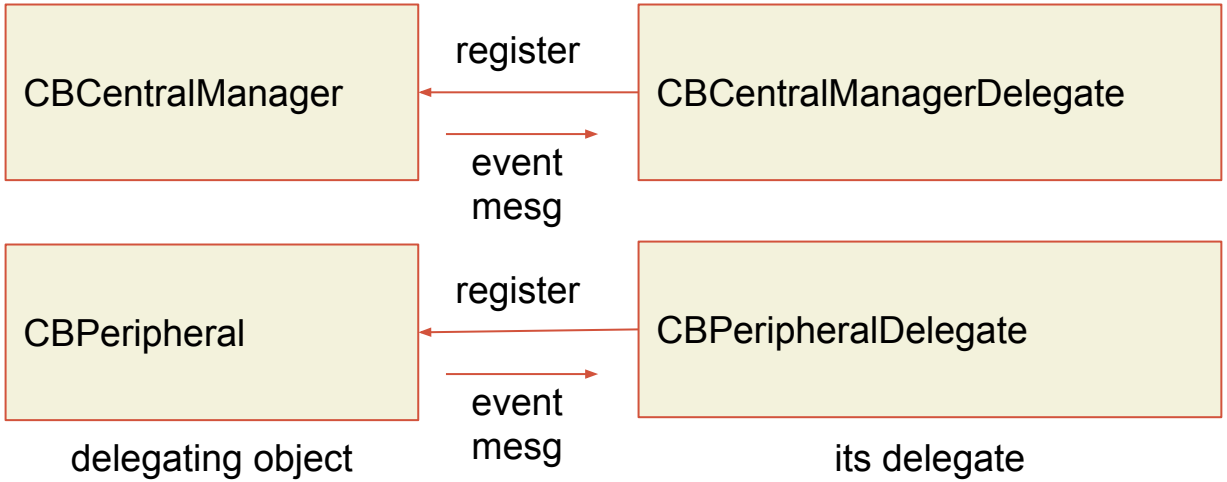
Ref. Apple, Inc.

Assignment#1-1

Event-driven programming

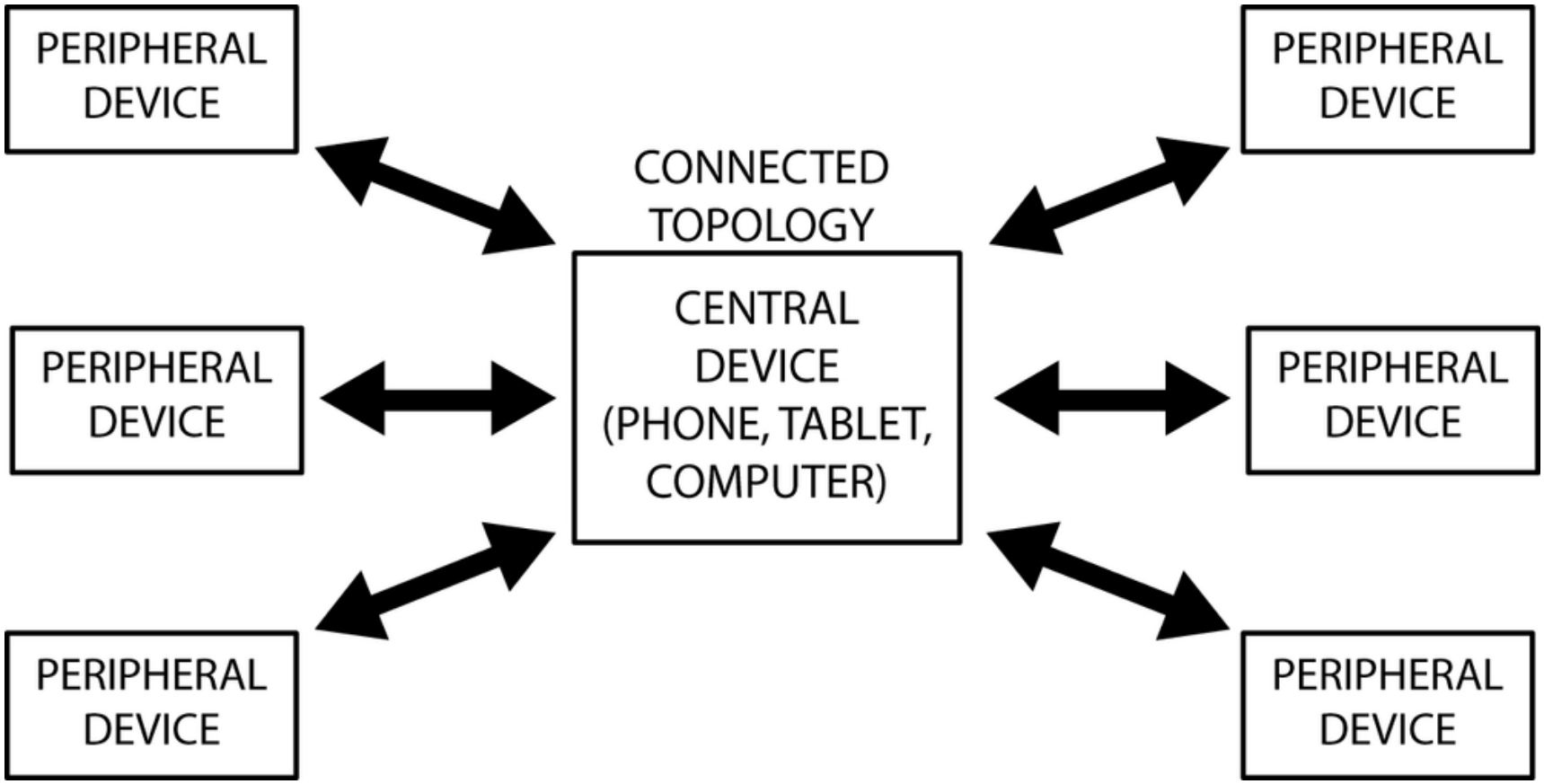
Explicitly called by user
Perform actions, Send mesg to delegate
User uses APIs only

Automatically called
Event handler, Callback functions
User implemented



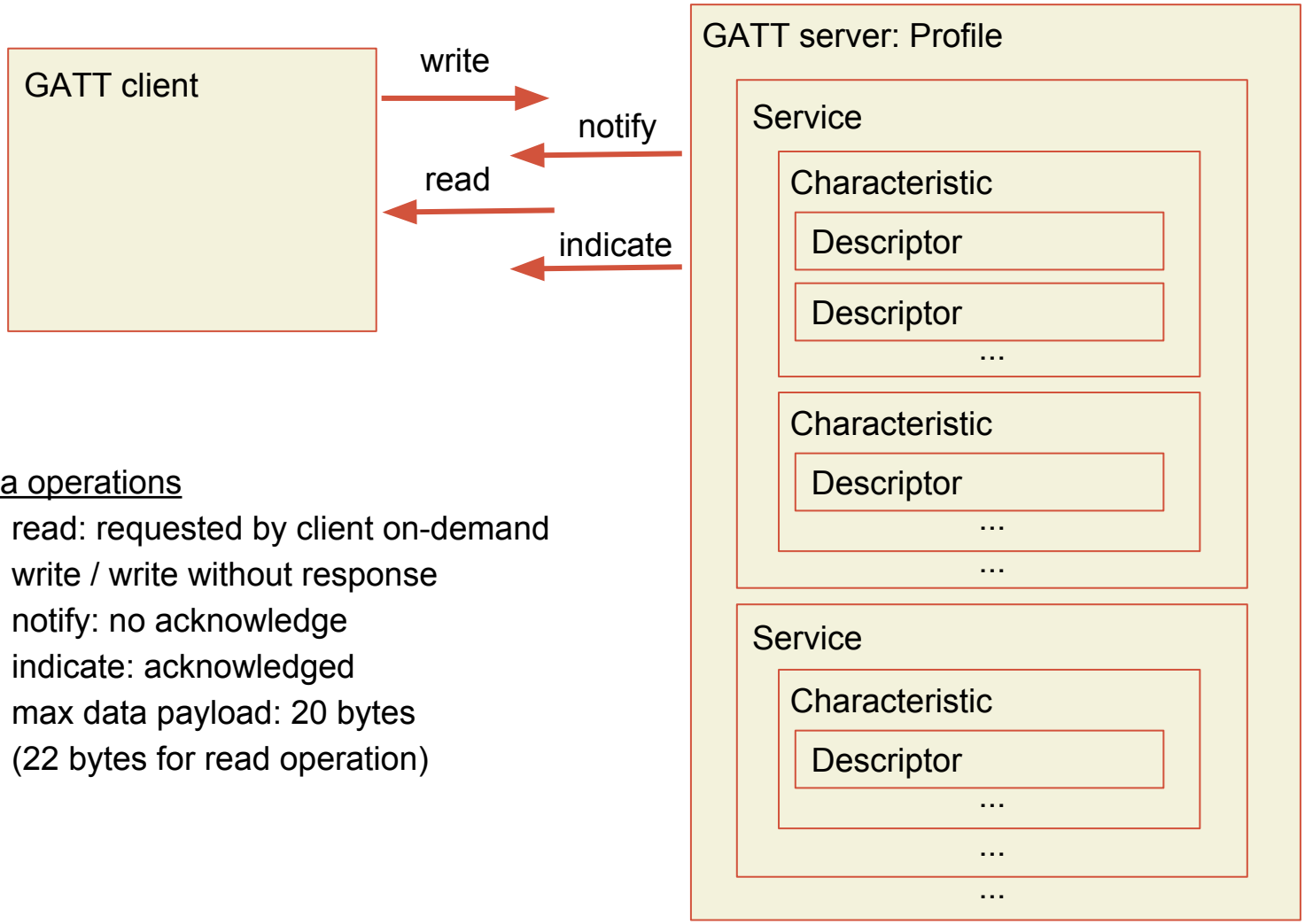
Note) CBCentralManager::centralManagerDidUpdateState_(self, manager) will be called automatically at the start.
Ex) By calling CBCentralManager::scanForPeripheralsWithServices_options_([Ads], None), CBCentralMangerDelegate::centralManager_didDiscoverPeripheral_advertisementData_RSSI_(self, manager, peripheral, data, rssi) will be called automatically.
Ex) By calling CBPeripheral::peripheral.discoverServices_([Name, Info, Sensors, Effectors]) CBPeripheralDelegate::peripheral_didDiscoverServices_(self, peripheral, error) will be called automatically.

Role: Central - Peripheral



Ref. “Getting started with Bluetooth Low Energy” by Townsend, Davidson & Akiba, O’Reilly

GATT: Data Transfer Methods



4 data operations

- read: requested by client on-demand
- write / write without response
- notify: no acknowledge
- indicate: acknowledged
- max data payload: 20 bytes
  (22 bytes for read operation)

Assignment#1-1: Due 10/01/2015

Hints:
In OS X, CentralManager == host/client/computer, Peripheral == slave/server/device(Hamster).

File: "hamsterAPI_ref.py"
Look at the required methods for delegates and available class methods in "hamsterAPI_ref.py".
Notice that this file is not a correct python file. This file lists relevant APIs that you should use.
--You need to implement the methods for both CBCentralManagerDelegate and CBPeripheralDelegate protocols.
--And you need to use CBCentralManager class methods and CBPeripheral class methods in delegate methods.
***** Note: this is event driven programming such that once you call the CentralManager class methods or Peripheral class methods, corresponding delegate methods are being called automatically.
Ex) By calling CBCentralManager::scanForPeripheralsWithServices_options_([Ads], None), CBCentralMangerDelegate::centralManager_didDiscoverPeripheral_advertisementData_RSSI_(self, manager, peripheral, data, rssi) will be called automatically.

Reference:
Mac OS X's Core Bluetooth Framework Reference
https://developer.apple.com/library/prerelease/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework/index.html#//apple_ref/doc/uid/TP40011295
PyObjC that is a bridge between Python and Objective-C.
https://pythonhosted.org/pyobjc/core/intro.html

What is an API?

API: Application Programming Interface.

Application

software that we use

Programming

process of creating software

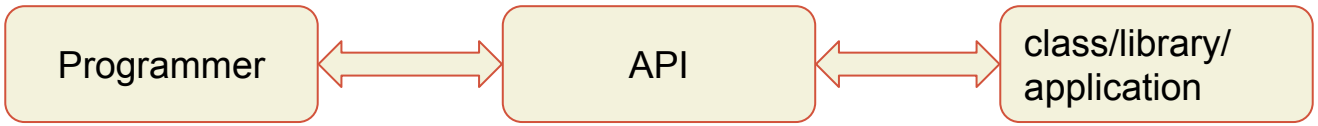
Interface

a common tool that enables two applications or programs to communicate with one another

API Call

When specific information from an API is needed, a program needs to call that API (make an API call).

API: a way for programmers to communicate with class/library/application.



Interface vs Implementation

- API describes
  - what a class/library/application does (interface of the class)
  - not how a class does it (implementation of the class)
  - Encapsulation: you have to implement API
- Interface
  - ex) CBCentralManager, CBCentralManagerDelegate
  - all we need to know to use a class
  - how we interact with a class
    - what methods to call
    - what they will return
- Implementation
  - ex) CBCentralManager: we don't care
  - ex) CBCentralManagerDelegate: we implement
  - using specific solutions for the given problems

Why is API Design Important?

- APIs can be among a company's greatest assets
  - Customers invest heavily: buying, writing, learning
  - Cost to stop using an API can be prohibitive
  - Successful public APIs capture customers
- Can also be among company's greatest liabilities
  - Bad APIs result in unending stream of support calls
- Public APIs are forever - one chance to get it right

Ref: “How to Design a Good API and Why it Matters” by Joshua Bloch, Google, Inc.

Why is API Design Important to You?

- If you program, you are an API designer
  - Good code is modular – each module has an API
- Useful modules tend to get reused
  - Once module has users, can't change API at will
  - Good reusable modules are corporate assets
- Thinking in terms of APIs improves code quality

Ref: "How to Design a Good API and Why it Matters" by Joshua Bloch, Google, Inc.

Characteristics of a Good API

- Easy to learn
- Easy to use, even without documentation
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to extend
- Appropriate to audience

Ref: “How to Design a Good API and Why it Matters” by Joshua Bloch, Google, Inc.

API: General Principal

API Should Do One Thing and Do it Well

- Functionality should be easy to explain
  - If it's hard to name, that's generally a bad sign
  - Good names drive development
  - Be amenable to splitting and merging modules
- Can be easier and more rewarding to build something more general
- Conceptual weight more important than bulk
- Implementation should not impact API
  - Don't let implementation details “leak” into API
- When in doubt leave it out
  - Functionality, classes, methods, parameters, etc.
  - You can always add, but you can never remove

Ref: “How to Design a Good API and Why it Matters” by Joshua Bloch, Google, Inc.

Naming Convention

In computer programming, a **naming convention** is a set of rules for choosing the character sequence to be used for identifiers which denote variables, types, functions, and other entities in source code and documentation.

Reasons for using a naming convention (as opposed to allowing programmers to choose any character sequence) include the following:

- to reduce the effort needed to read and understand source code;[1]
- to enable code reviews be able to focus on more important issues than arguing over syntax and naming standards.
- to enable code quality review tools be able to focus their reporting mainly on significant issues other than syntax and style preferences.
- to enhance source code appearance (for example, by disallowing overly long names or unclear abbreviations).

Ref. Wikipedia

Naming Convention: Python

Python.org

https://www.python.org/dev/peps/pep-0008/#naming-conventions

Class Names

Class names should normally use the CapWords convention.

Function Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Method Names and Instance Variables

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

Use one leading underscore only for non-public methods and instance variables.

Ref. Python.org

Hamster: Robot API

- Accessor
  - only retrieve information
  - read values from Sensors packet (20 bytes)
  - start with “get_”

- Mutator
  - modify values or state
  - write values to Effectors packet (20 bytes)
  - start with “set_”

Bytes and Bits

1 Byte = 8 bits (1 bit = binary number):

- smallest addressable unit of memory
- usually expressed in hex since 1 byte = 8 bits = 2^8 numbers
- a single character of text
- [0, 255] if positive value only
- [-128, 127] in Two's complement

ex) 0x00 = 0000 0000 = 0

0x0F = 0000 1111 = 2^0 + 2^1 + 2^2 + 2^3 = 2^4 - 1 = 15

0xFF = 1111 1111 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7
= 2^8 - 1 = 255 or -1 in Two's complement

0xF0 = 1111 0000 = 2^8 - 2^4 = 240 or -16 in Two's complement

0x80 = 1000 0000 = 2^7 = 128 or -128 in Two's complement

0x7F = 0111 1111 = 2^7 - 1 = 127

Bitwise operators and masks

Operators: <<, >>, &, |, ^, ~

x << y: left shift (same as multiplying x by 2^y)

Shift x's bits to the left by y places (and new bits on the right-hand-side are zeros).

x >> y: right shift (same as dividing x by 2^y if most significant bit is 0)

Shift x's bits to the right by y places (and new bits on the left-hand-side are the same as the most significant bit before the shift).

x & y: bitwise and

Each bit of output is 1 if AND of corresponding bits in x and y is 1, otherwise 0.

x | y: bitwise or

Each bit of output is 1 if OR of corresponding bits in x and y is 1, otherwise 0.

x ^ y: bitwise xor

Each bit of output is 1 if corresponding bits in x and y are different, otherwise 0.

~x: not / one's complement (same as -x-1)

Invert each bit of x such that 0 → 1 and 1 → 0.

Masks: Using a mask, multiple bits can be set either on, off or inverted in a single bitwise operation.

Lower 4 bit mask: 0x0f

a = 0x1A (26) = 0001 1010

a & 0x0f = 1010 (10)

Bitwise operations: example

```
#!/usr/bin/python

a = 60             # 60 = 0011 1100
b = 13             # 13 = 0000 1101
c = 0

#bitwise operators

c = a & b        # 12 = 0000 1100

c = a | b        # 61 = 0011 1101

c = a ^ b        # 49 = 0011 0001

c = ~a           # -61 = 1100 0011

c = a >> 2       # 15 = a/4 = 0000 1111

c = a << 2       # 240(-16) = a*4 = 1111 0000

c = c >> 4       # 255(-1) = 1111 1111

#masks

c = (a & 0x0f)       # value of lower 4 bits (bits 0~3): 12(c) = 1100

c = (a >> 4) & 0x0f  # value of upper 4 bits (bits 4~7): 3 = 0011
```

Two's complement

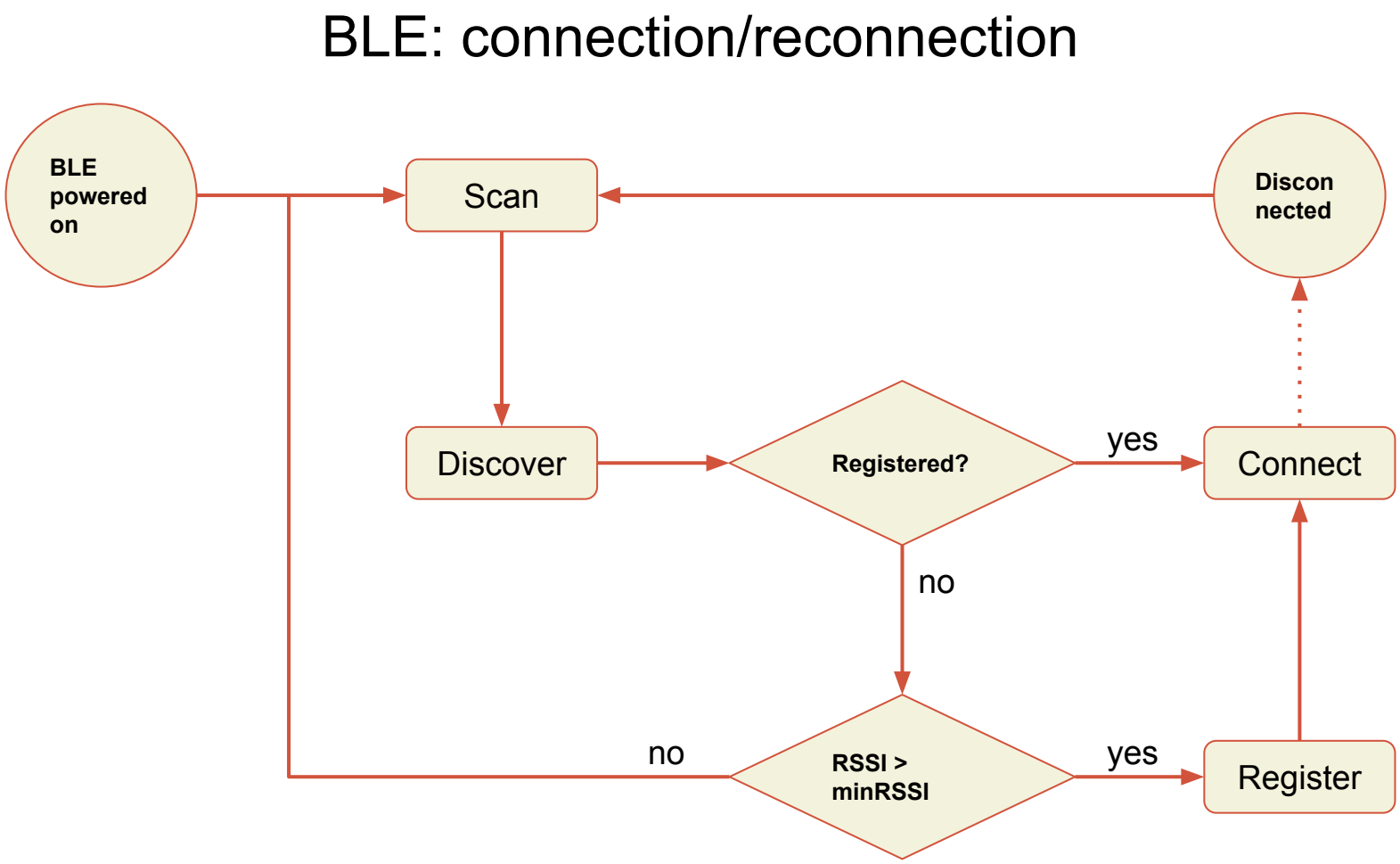
Conversion from Two's complement:

1. Invert binary number of a number with most significant bit = 1
   0xF0 (240) = 1111 0000 → 0000 1111
2. Add 1
   0000 1111 → 0001 0000 = 16
3. negate value from #2
   -16 = 0xF0

Conversion to Two's complement:

1. Get an absolute value of a negative number
   -16 → 16 = 0001 0000
2. Invert binary number of value from #1
   0001 0000 → 1110 1111
3. Add 1
   1110 1111 → 1111 0000 = 0xF0 (2^7 + 2^6 + 2^5 + 2^4 = 240)

Assignment#1-2



Assignment#1-2: Due 10/06/2015

Reference:
Hamster Manual
http://web.stanford.edu/class/cs123/materials/Hamster_Manual.pdf

Style Guide for Python Code: Naming Conventions by www.python.org
https://www.python.org/dev/peps/pep-0008/#naming-conventions

Reference and Reading

"Getting started with Bluetooth Low Energy" by Townsend, Davidson & Akiba, O'Reilly
https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/cover.html

"How to Design a Good API and Why it Matters" by Joshua Bloch, Google, Inc.
http://lcsd05.cs.tamu.edu/slides/keynote.pdf

"Core Bluetooth Programming Guide" by Apple, Inc. (Introduction, Core Bluetooth Overview, Performing Common Central Role Tasks, Best Practices for Interacting with a Remote Peripheral Device)

"Core Bluetooth Framework Reference" by Apple, Inc. (Classes: CBCentralManager/CBPeripheral, Protocols: CBCentralManagerDelegate/CBPeripheralDelegate)

Style Guide for Python Code: Naming Conventions by www.python.org
https://www.python.org/dev/peps/pep-0008/#naming-conventions

Hamster Manual
http://web.stanford.edu/class/cs123/materials/Hamster_Manual.pdf