CS123 - Structure & Mobile

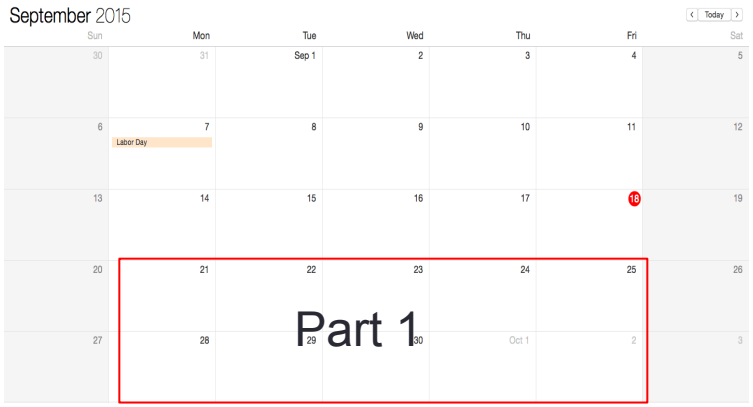
Programming Your Personal Robot

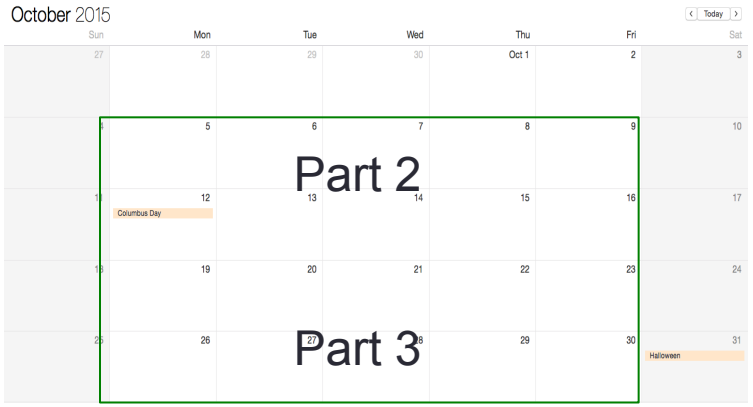
Kyong-Sok “KC” Chang, David Zhu
Fall 2015-16

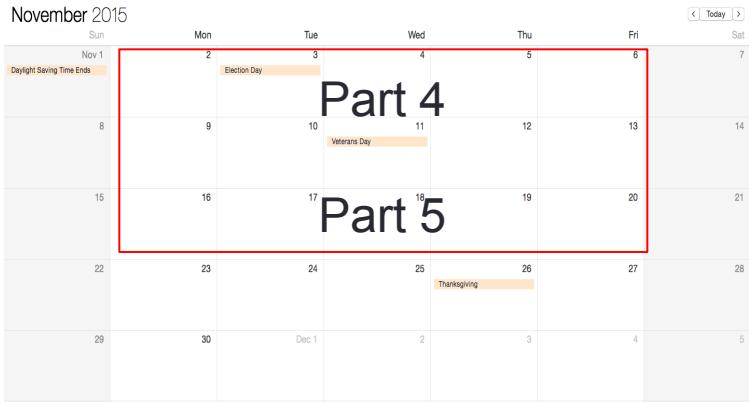
Logistics

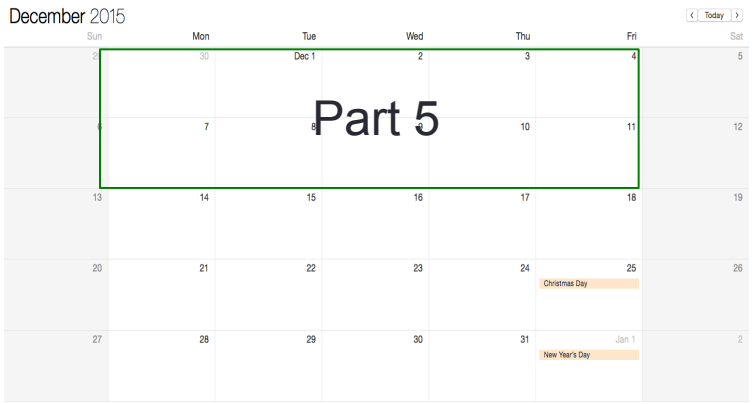
- Getting your own Hamster (and USB BLE dongle)
  - Sign-up sheet
- Programming environment after Assignment#1-1,1-2
  - OS: OS X, Windows, Linux
- TA sessions (office hours): this week
  - Location: Gates B21
  - Time: M:2~4pm, Tu:6~8pm, W:12:30-2:30pm, Th:6:30~8:30pm
- Lab reserved for CS123: this week
  - MTuW: 12~6pm, ThF: 24hr @ Gates B21
- Mac laptop from the Lathrop Tech Lounge: 24hr each
- My office hours
  - Friday(10/2), Monday(10/5): 12:30-2:00pm @ Gates B21

Calendar









KC
Teaching

David
Teaching

USB-BLE Dongle

Hamster: using Python via USB dongle for Windows, Linux, OS X

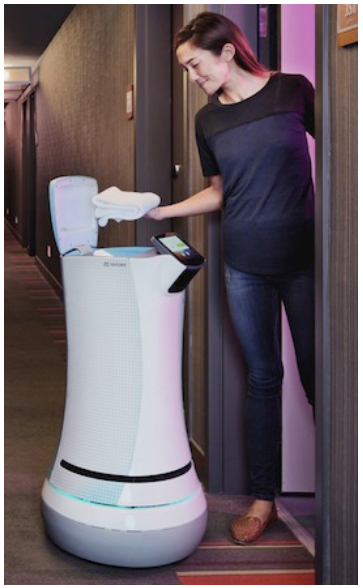
1. Install: USB dongle driver
Plug in the USB dongle to the computer (For Windows, this will automatically install the driver).
Otherwise, refer to
https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx
2. Install: Python 2.7 (This will also install “pip”)
https://www.python.org/downloads/

- For Windows, Make sure to add python path to the PATH environment variable
    1. During the installation of Python27, choose “Add python.exe to Path” in the install setup menu (at the bottom of the install options list).
    2. Or After the installation, manually append “C:\Python27\;C:\Python27\Scripts;” to the beginning of the PATH environment variable
- For Linux and OS X, Python should be pre-installed.

3. Install: pySerial 2.7
In the command window(cmd.exe) or in the terminal(Linux, OS X)
> pip install pyserial
or in the terminal(Linux or OS X)
> sudo easy_install -U pyserial
If it doesn’t work, refer to the “pip” site:
https://pip.pypa.io/en/latest/installing.html
4. Connect: Hamster with USB dongle
First time, turn on the Hamster and bring it close (10~20 cm) to the USB dongle
From then on, turn on the Hamster near the USB dongle (BTL range: ~ 10 m)
Before connection, BTL indicator on Hamster and on USB dongle will be blinking blue.
After connection, BTL indicator on Hamster and on USB dongle will be solid blue.

Future: Delivery/Logistics Robots?

Photo: Joel Eden Photography/Kiva Systems



Outline

- Lecture #03
  - Bytes and Bits
  - Bitwise operators and masks
  - Two's complement
- BLE Mobile Device
  - Network Topology
  - Android
- Structure: Python Project
  - Module vs Package
  - Importing
  - Utility functions
- Assignment#1-1
- Assignment#1-2

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:

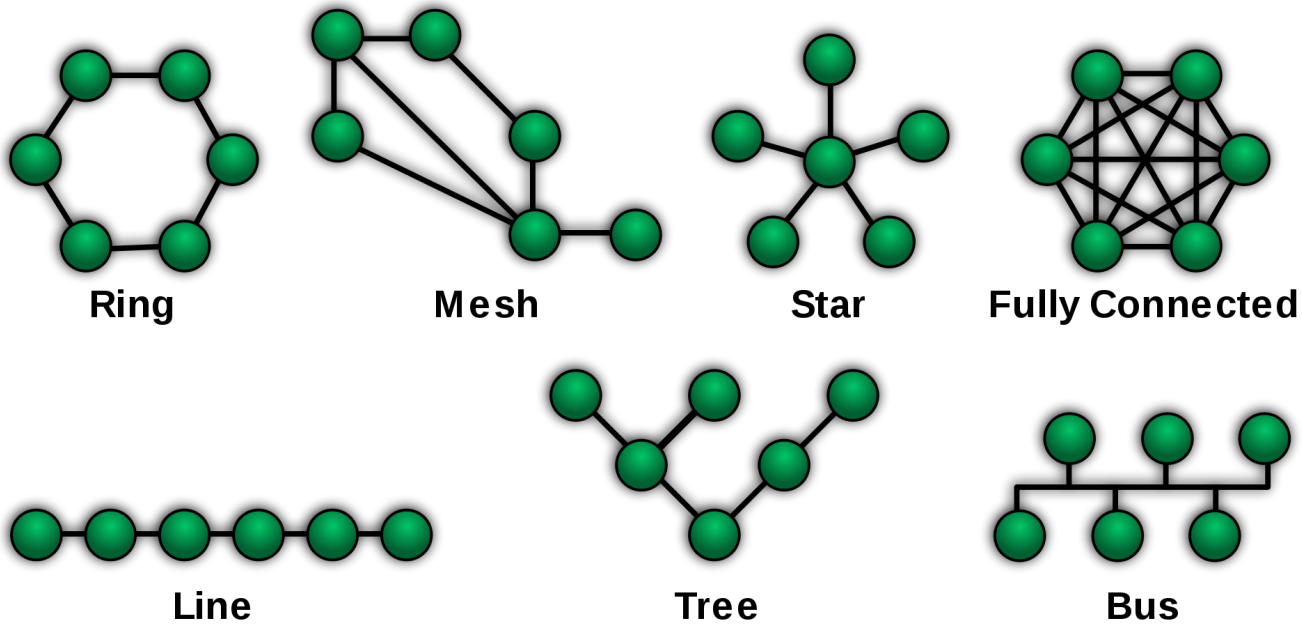
1. Invert binary number of a number with most significant bit = 1
   0xF0 (240) = 1111 0000 → 0000 1111
2. Add 1
   0000 1111 → 0001 0000 = 16
3. negate value from #2
   -16 = 0xF0

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)

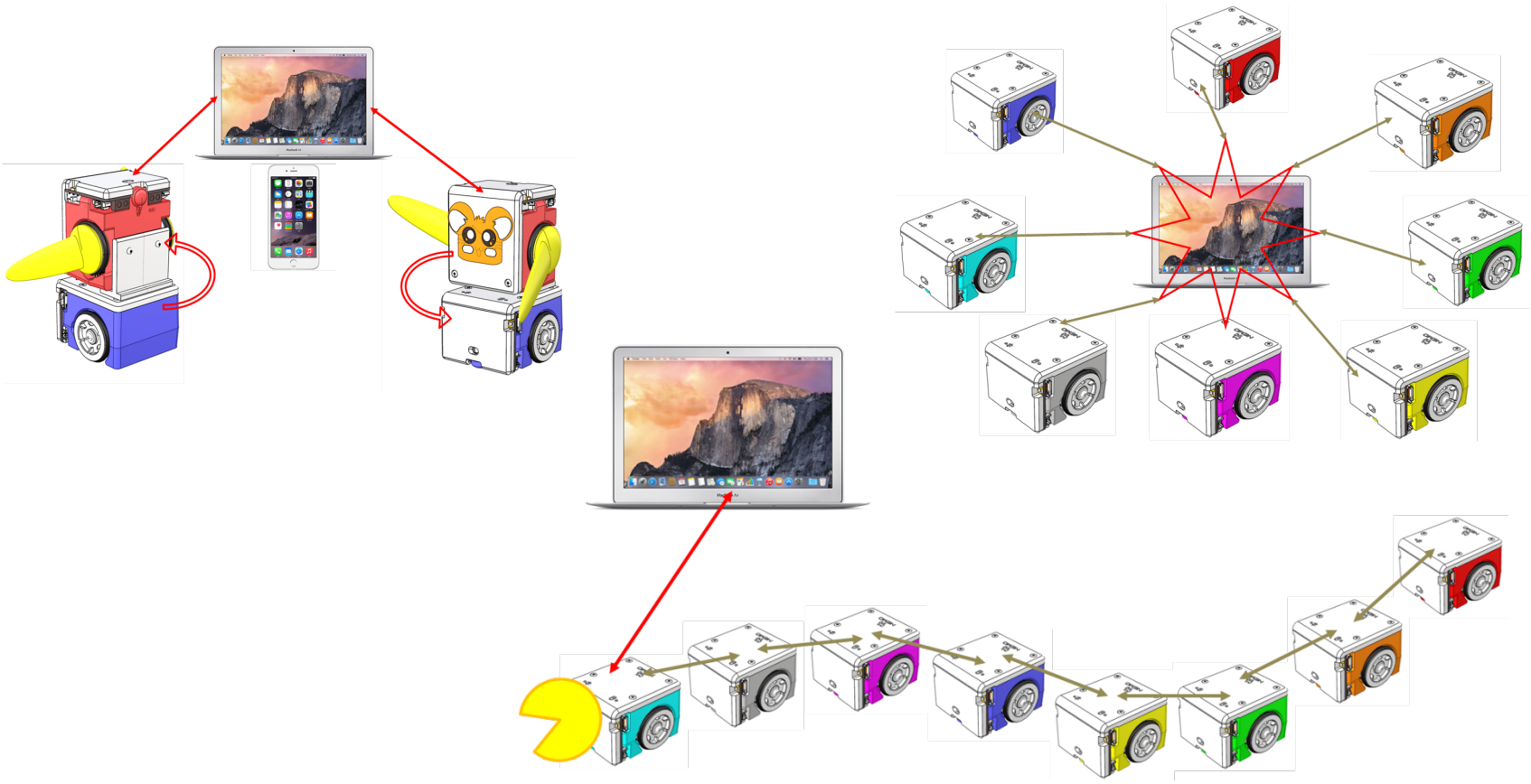
BLE Device: Network Topology

Common Topologies



Hamster: Network Topology

Hamster: Star (8), Tree (3), Daisy chain



BLE Mobile Device: Android

Bluetooth Low Energy (Android API Guides)

Android 4.3 (API Level 18) introduces built-in platform support for Bluetooth Low Energy in the *central role*

1. **Roles and Responsibilities**
   a. Central vs. peripheral : (advertisement) : scanning vs advertising
   b. GATT client vs. GATT server : (data) : requesting vs providing
2. **BLE Permissions**
   a. Declare the Bluetooth permission(s) in your application manifest file (xml file)
   b. `<uses-permission android:name="android.permission.BLUETOOTH"/>`
   c. `<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>`
   d. `<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>`
3. **Setting Up BLE**
   a. Get the BluetoothAdapter (BLE host hardware): CBCentralManager
   b. Check isEnabled(). Enable Bluetooth
4. **Finding BLE Devices**
   a. To find BLE devices, you use the startLeScan() method.
   b. This method takes a BluetoothAdapter.LeScanCallback as a parameter. You must implement this callback.
   c. **public abstract void onLeScan (BluetoothDevice device, int rssi, byte[] scanRecord)**
   d. **BluetoothDevice** (BLE device)
      i. connection
      ii. info

Ref. android.com by Google

BLE Mobile Device: Android

5. **Connecting to a GATT Server**
   a. To connect to a GATT server on a BLE device, you use the connectGatt() method.
   b. mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
   c. **BluetoothGatt**:
      i. CBPeripheral
      ii. discoverServices(), writeCharacteristic(), setCharacteristicNotification(), etc
   d. BluetoothGattCallback:
      i. CBPeripheralDelegate
      ii. onServiceDiscovered(), onCharacteristicWrite(), onCharacteristicChanged() , etc
6. **Reading/writing BLE Attributes (receiving GATT notification)**
   a. **BluetoothGattService**
   b. **BluetoothGattCharacteristic**
   c. **BluetoothGattDescriptor**
7. **Receiving GATT Notifications**
   a. set a notification for a characteristic, using the setCharacteristicNotification()
   b. Once notifications are enabled for a characteristic, an onCharacteristicChanged() callback is triggered if the characteristic changes on the remote device:
8. **Closing the Client App**
   a. close() : release resources

Ref. android.com by Google

Structure: Python Project

- "Structure" means
  - making clean and effective code
  - whose logic and dependencies are clear
  - and files and folders are organized effectively in the filesystem
- Low-level layer
  - low-level manipulation of data
  - RobotDelegate
- Interface layer
  - interfacing with user actions
  - needs to import low-level file
  - RobotAPI
- Main application
  - needs to import interface file
  - hamster_class_api.py

Structure: Module vs Package

- Module
  - provides abstraction layer
  - namespace
  - a file
  - group of related classes

- Package
  - extension of module mechanism to a directory
  - namespace
  - a directory
  - group of related modules
  - __init__.py file
    - executed first when imported
    - gathers all package-wide definitions
    - empty file: normal practice if there is no

Structure: Importing

- Importing modules/packages
  - import
  - import … as
  - from … import
- Example
  - Very bad

    ```
    from math import *
    [...]
    x = sqrt(9) # Is sqrt part of math? A built-in? Defined above?
    ```
  - Better

    ```
    from math import sqrt
    [...]
    x = sqrt(9) # sqrt may be part of math, if not redefined in between
    ```
  - Best

    ```
    import math
    [...]
    x = math.sqrt(9) # sqrt is visibly part of math's namespace
    ```

Structure: Utility functions

- Utility functions
  - Functions
  - Ex) Two's complement
  - module vs package (sub-package if subdirectory)
  - who uses?

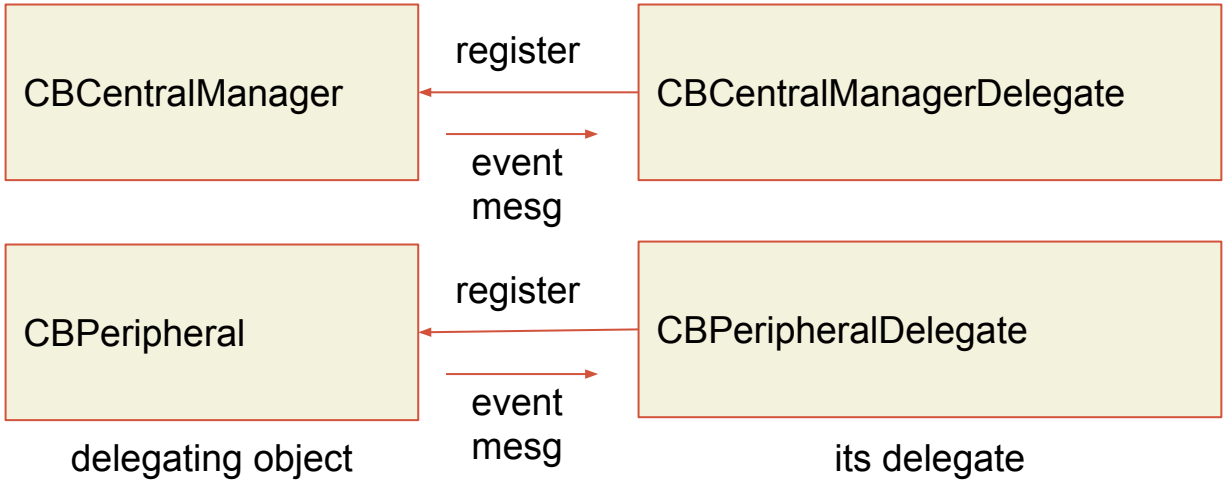
- "The Hitchhiker's Guide to Python!" by A Kenneth Reitz Project
  - http://docs.python-guide.org/en/latest/
  - (Structuring Your Project)

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

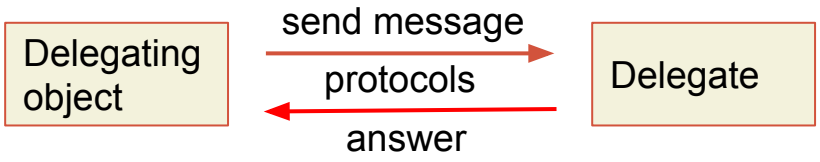


Note) CBCentralManager::centralManagerDidUpdateState_(self, manager) will be called automatically at the start.
Ex) By calling CBCentralManager::scanForPeripheralsWithServices_options_([Ads], None), CBCentralMangerDelegate::centralManager_didDiscoverPeripheral_advertisementData_RSSI_(self, manager, peripheral, 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.

Delegation vs Protocol

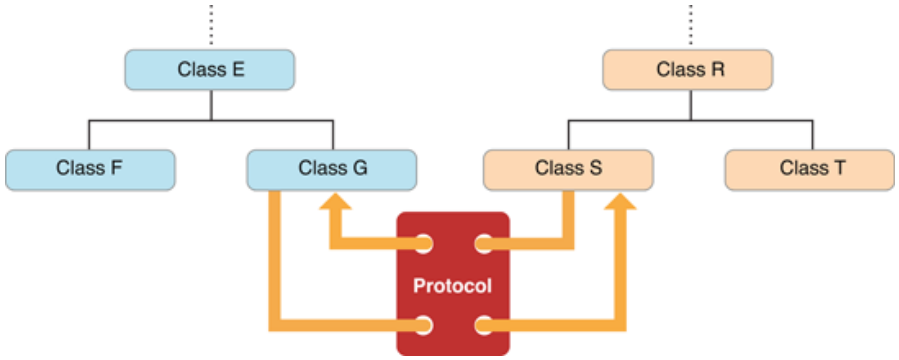
Delegation: Acting on Behalf of Another Object

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.



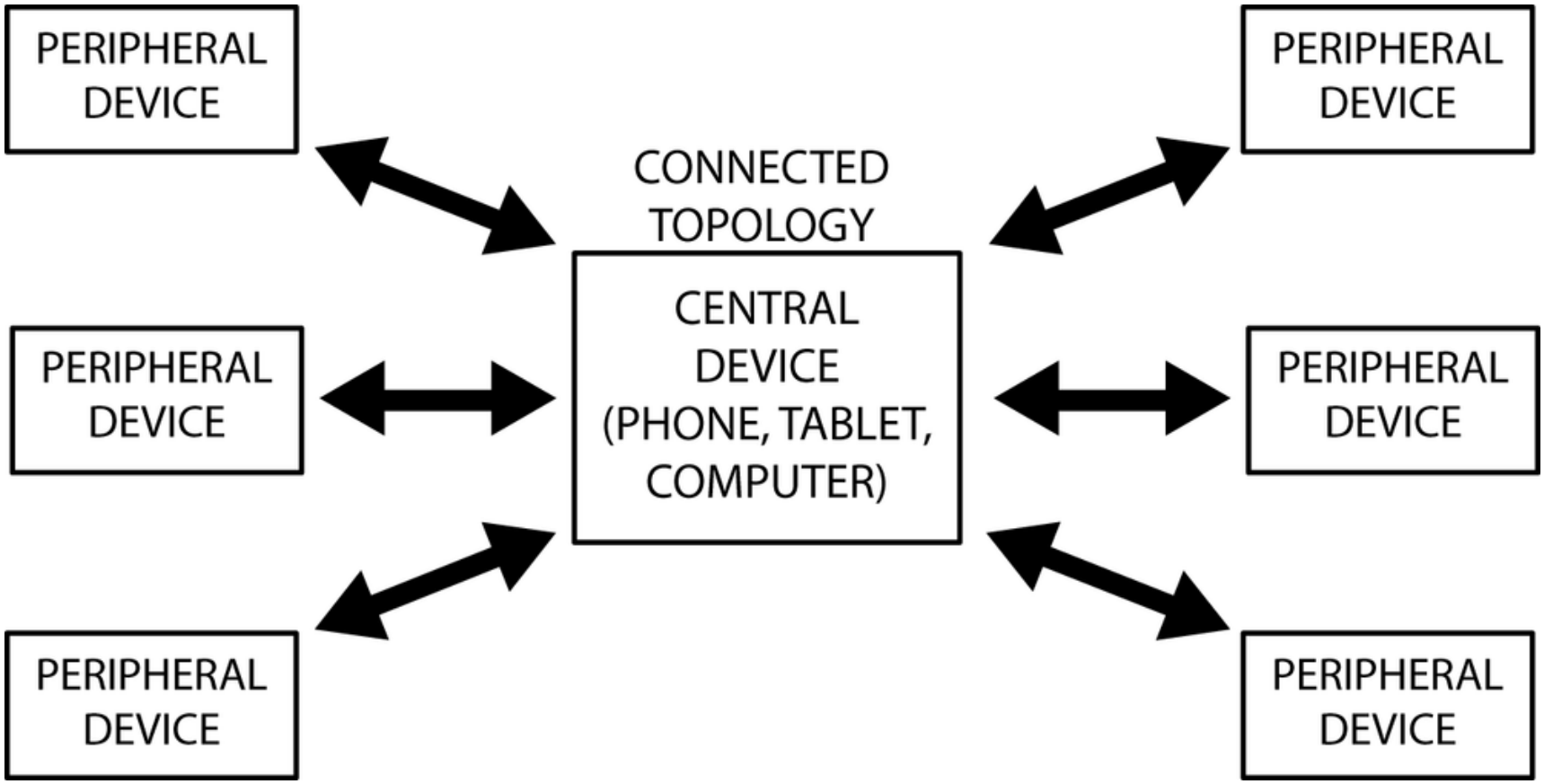
Protocol: Enabling Communication Between Objects Not Related by Inheritance

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.



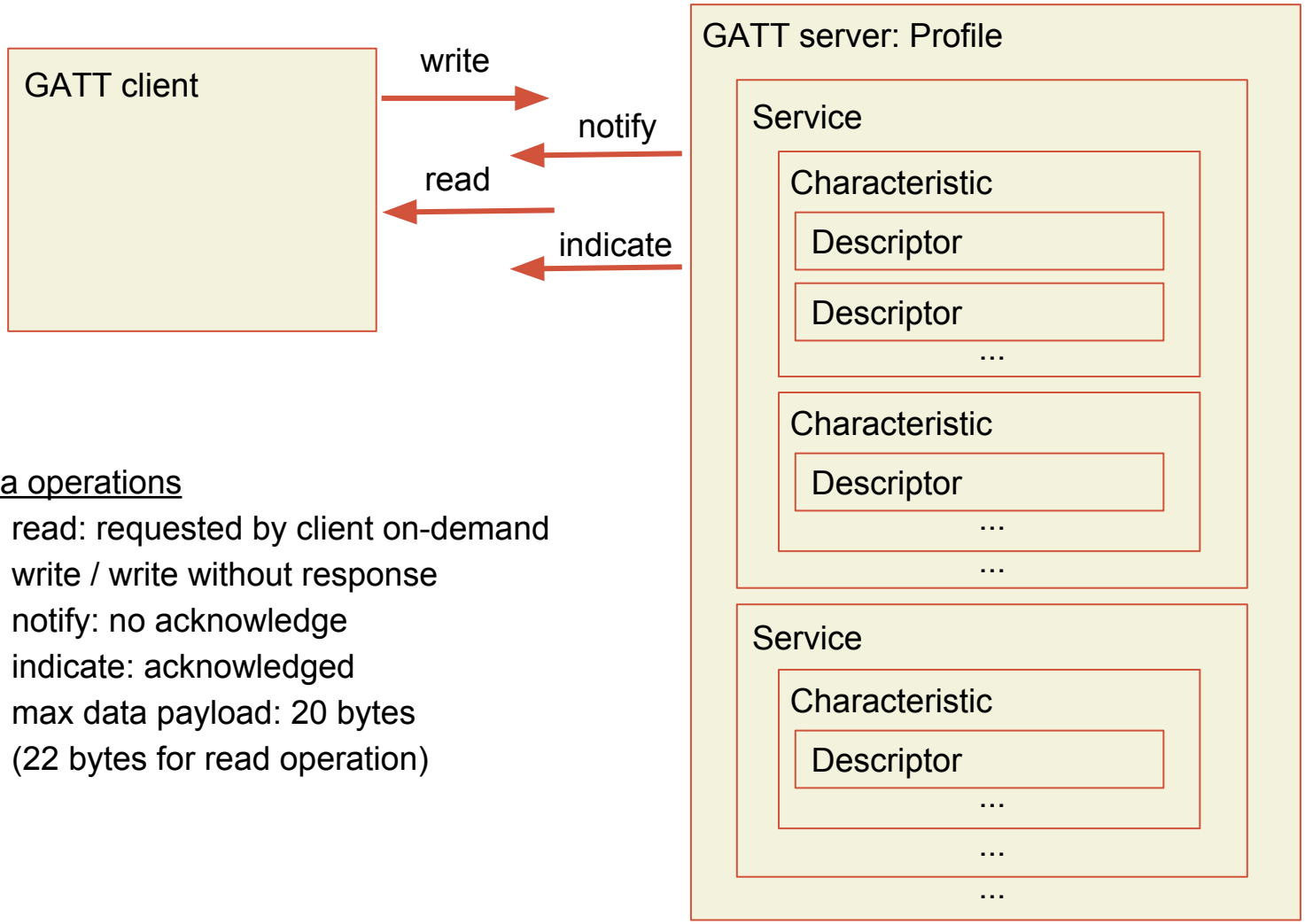
Ref. Apple, Inc.

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)

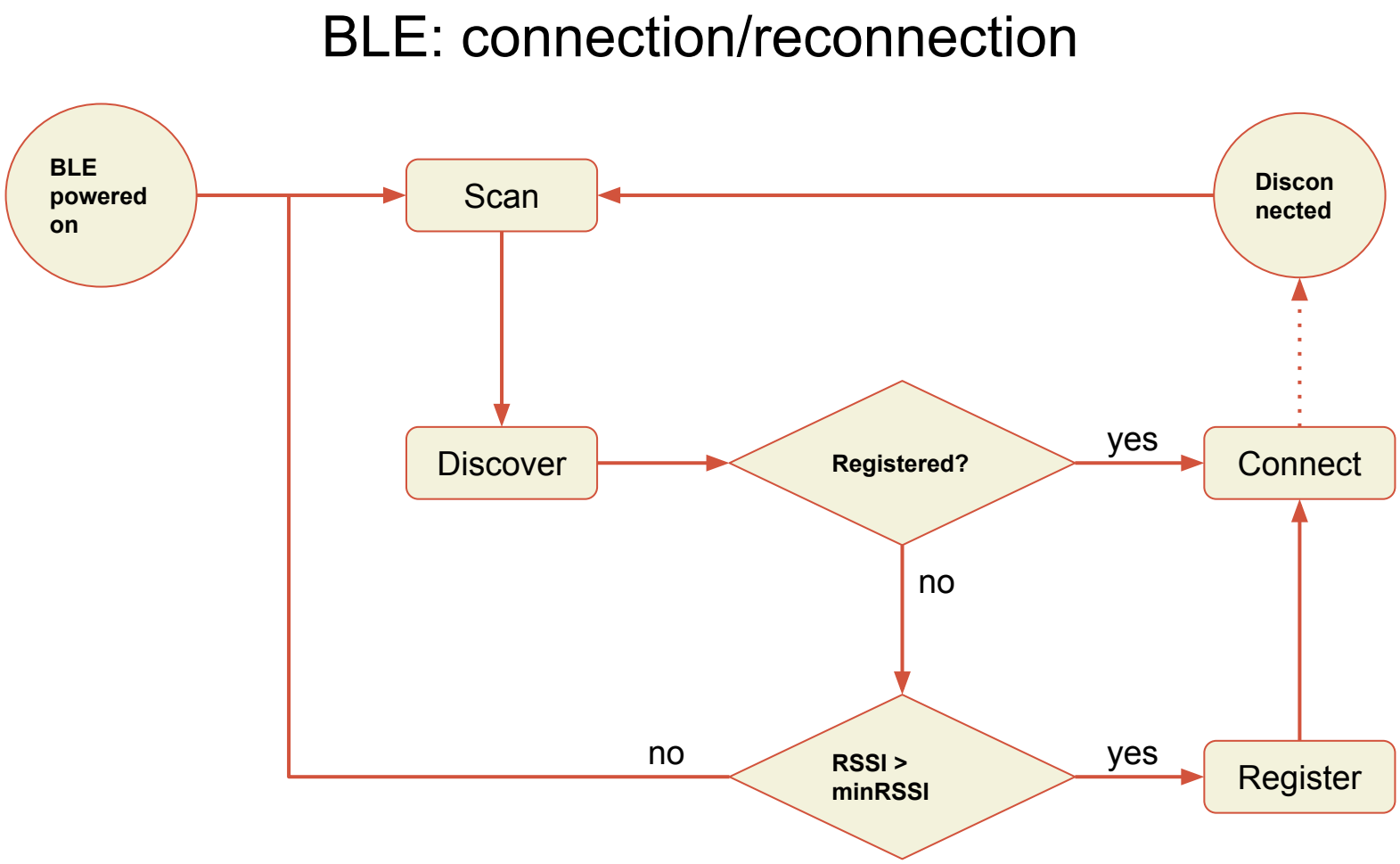
Assignment#1-1: Due 10/01/2015

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:
"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)
"An Introduction to PyObjC"
https://pythonhosted.org/pyobjc/core/intro.html

Assignment#1-2



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_”

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

Reference:

"Hamster Manual" by Kre8 Technology, Inc.
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

"The Hitchhiker's Guide to Python!" (Structuring Your Project)
http://docs.python-guide.org/en/latest/

Reference and Reading

"Getting started with Bluetooth Low Energy" by Townsend, Davidson & Akiba, O'Reilly

"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)

"An Introduction to PyObjC" by Python Software Foundation

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

"Style Guide for Python Code" by Python Software Foundation (Naming Conventions)

"The Hitchhiker's Guide to Python!" by A Kenneth Reitz Project (Structuring Your Project)

"Hamster Manual" by Kre8 Technology, Inc.

"**Bluetooth Low Energy (Android API Guides)" by android.com, Google, Inc.**