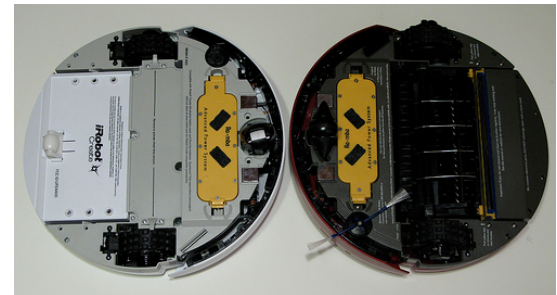
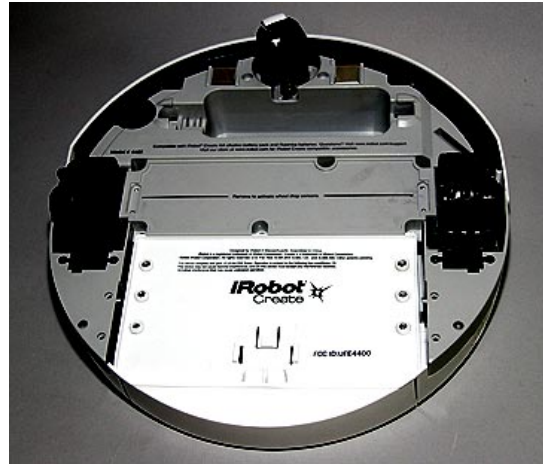


The *Create* robot, by iRobot

- Same as the 4000 series of the Roomba vacuum cleaner, but:
 - Without the vacuum
 - With a port that gives easy access to the robot's "brain"



Students: These slides present important background information, but are NOT something to memorize

Robot hardware

Robots generally have:

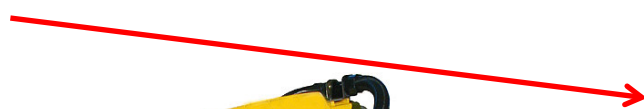
- **Effectors** – allow the robot to take an action, e.g.:



- **Move**



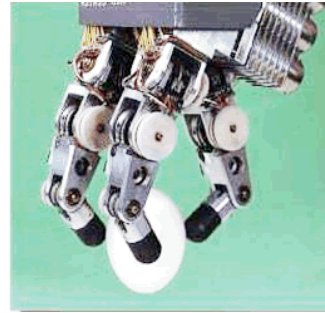
- **Grasp**



- **Touch**



- **Talk**

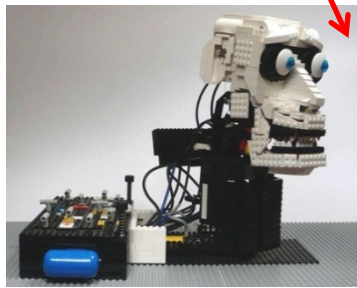
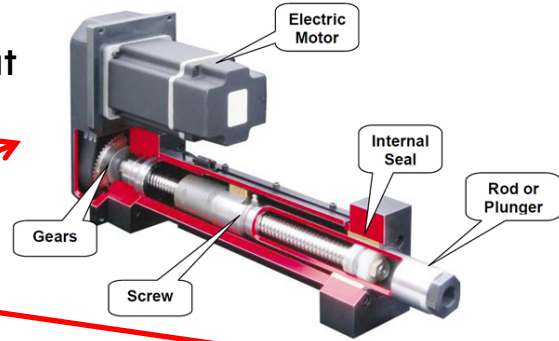
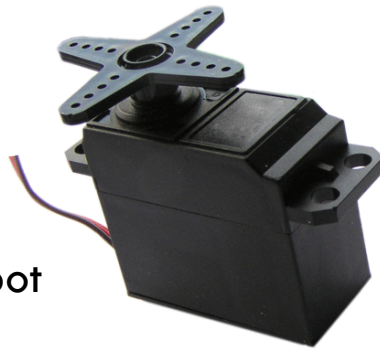


Robot hardware

Robots generally have:

- **Effectors** – allow the robot to take an action
- **Actuators** – mechanisms that drive the effectors, e.g.:

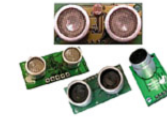
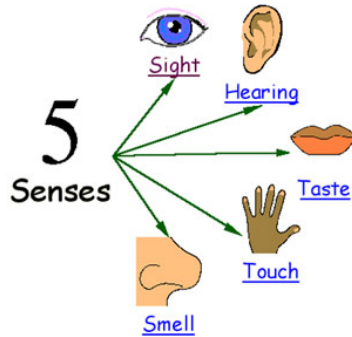
- Motors / servos
- Hydraulics
- Pneumatics



Robot hardware

Robots generally have:

- **Effectors** – allow the robot to take an action
- **Actuators** – mechanisms that drive the effectors
- **Sensors** – allow the robot to know (sense) things about the world



Ultrasonic Range Finders



Force Sensors



Inertia Measurement Units



Localization



Encoders and Disks



Current and Voltage Sensors



Infrared and Light Sensors



Accelerometers



Inclination & Tilt Sensors



Cameras and Vision Sensors



Linear and Rotary Resistors



Temperature and Humidity Sensors



Stretch and Bend Sensors



Gyroscopes



RFID



Contact and Proximity Sensors



Magnetic Sensors / Compass



Thermal Array Sensors

iRobot Create hardware

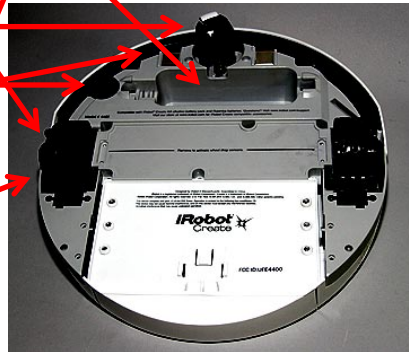
- **Effectors** – 3 wheels, 1 speaker, 3 lights, output ports that send signals

- **Actuators:**

- 2 independent motors
- Battery power

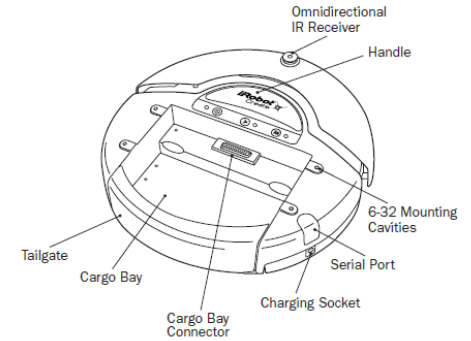
- **Sensors:**

- 2 touch (bump)
- 3 buttons
- 3 wheel-drop sensors
- 4 active IR cliff sensors
- 1 passive IR receiver
- 2 wheel encoders
- And more!

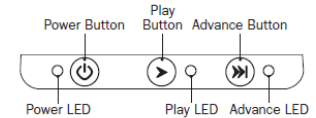


Anatomy

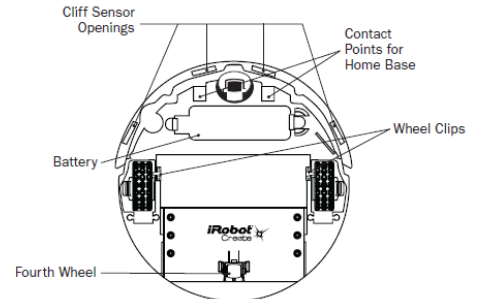
Top View



Buttons and Lights



Bottom View





Your Python program uses the `create` library. It supplies functions that send commands:

- to your laptop's Bluetooth radio,
- then to the BAM on the Create,
- then to the Create's controller.

Ditto for the reverse path.



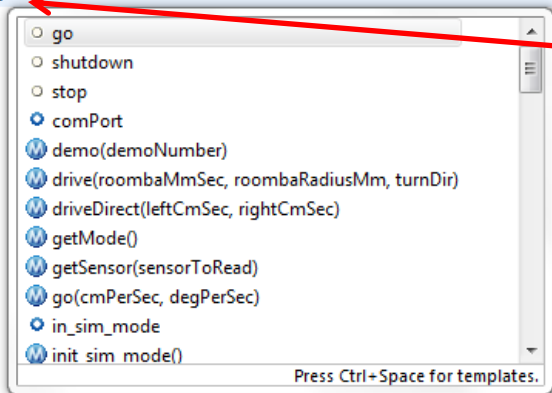
```
import create
import time

def main():
    """ Constructs a robot and makes it move, then shut down. """
    port = 13 # Port your Bluetooth connected to (more in class on this)
    robot = create.Create(port) # Construct and connect

    robot.go(10, 90) # Go forwards 10 cm/sec, spin 90 deg/sec counterclockwise
    time.sleep(4.0) # for 4 seconds
    robot.stop() # then stop

    robot.shutdown() # always end with this
```

```
robot = create.Create(port) # Construct and connect
robot.
```



As usual, *use the dot trick* to learn the methods in the Create class. This slide highlights a few particularly useful methods.

- driveDirect(leftCmSec, rightCmSec)
- getMode()
- getSensor(sensorToRead)
- go(cmPerSec, degPerSec)
- in_sim_mode
- init_sim_mode()
- maxSensorRetries
- playNote(noteNumber, duration, songNumber)
- playSong(noteList)
- playSongNumber(songNumber)
- read(bytes)
- reconnect(comPort)

- send(bytes1)
- sendIR(byteValue)
- sensorDataIsOK()
- ser
- serialLock
- setDigitalOutputs(digOut2, digOut1, digOut0)
- setLEDs(powerColor, powerIntensity, play, advance)
- setLowSideDrivers(driver2, driver1, driver0)
- setMaxSensorTimeout(newTimeout)
- setPWMLowSideDrivers(dutyCycle2, dutyCycle1, dutyCyc
- setSong(songNumber, noteList)

- shutdown()
- sim_host
- sim_port
- sim_sock
- start()
- startIR(byteValue)
- stop()
- stopIR()
- toFullMode()
- toSafeMode()
- waitAngle(degrees)

- waitAngle(degrees)
- waitDistance(centimeters)
- waitEvent(eventNumber)
- waitTime(seconds)
- _close()
- _closeSer()
- _getButtonBits(r)
- _getLower5Bits(r)
- _getOneBit(r)
- _getOneByteSigned(r)
- _getOneByteUnsigned(r)