What is OBD

On-board diagnostics (OBD) is a term referring to a vehicle's self-diagnostic and reporting capability. OBD systems give the vehicle owner or repair technician access to the status of the various vehicle sub-systems. The amount of diagnostic information available via OBD has varied widely since its introduction in the early 1980s versions of on-board vehicle computers. Early versions of OBD would simply illuminate a malfunction indicator light (MIL) or "idiot light" if a problem was detected, but would not provide any information as to the nature of the problem. Modern OBD implementations use a **standardized digital communications port** to provide real-time data in addition to a **standardized series of diagnostic trouble codes**, or DTCs, which allow a person to rapidly identify and remedy malfunctions within the vehicle.

OBD-II provides access to data from the engine control unit (ECU) and offers a valuable source of information when troubleshooting problems inside a vehicle. The SAE J1979 standard defines a method for requesting various diagnostic data and a list of standard parameters that might be available from the ECU. The various parameters that are available are addressed by **"parameter identification numbers" or PIDs** which are defined in J1979. For a list of basic PIDs, their definitions, and the formula to convert raw OBD-II output to meaningful diagnostic units, see OBD-II PIDs. **Manufacturers are not required to implement all PIDs listed in J1979** and they are allowed to include proprietary PIDs that are not listed. The PID request and data retrieval system gives access to real time performance data as well as flagged DTCs. For a list of generic OBD-II DTCs suggested by the SAE, see Table of OBD-II Codes. **Individual manufacturers often enhance the OBD-II code set with additional proprietary DTCs.**

1996: The OBD-II specification is made mandatory for all cars sold in the United States.

To see the definition of the speed PID (as defined by OBD2) see
https://en.wikipedia.org/wiki/OBD-II_PIDs (https://en.wikipedia.org/wiki/OBD-II_PIDs)

What you will need

An installation of python, see:
https://www.anaconda.com/products/distribution
(https://www.anaconda.com/products/distribution)

Required packages:

```
# The OBD package
pip install obd

# An emulator we can use to connect to (because bluetooth won't reach
the classroom)
# This emulator emulates a limited set of Toyota Auris Hybrid PIDs.
# For more information see the link below:
pip install ELM327-emulator
```

https://github.com/Ircama/ELM327-emulator (https://github.com/Ircama/ELM327-emulator)

Start by importing the OBD module and creating a connection to the OBD adapter

In [58]:
```
# Import the OBD module and create a connection to the OBD adapter
import obd

port = "/dev/pts/1"
connection = obd.OBD(port) # auto-connects to USB or RF port if no argument
```

Use one of the pre-defined commands in the python OBD module

To see the commands available in this package see:
https://github.com/brendan-w/python-OBD/blob/master/obd/commands.py
(https://github.com/brendan-w/python-OBD/blob/master/obd/commands.py)

In [ ]:
```
cmd = obd.commands.SPEED          # select an OBD command (sensor)

response = connection.query(cmd) # send the command, and parse the response

print(response.value)             # returns unit-bearing values thanks to Pi
print(response.value.to("mph"))  # user-friendly unit conversions
```

Note that the values coming from the emulator are random and will change between calls

The OBD package provides return types defined in the package Pint.

Pint is a Python package used to define and operate on physical quantities. It's types are a combination of numerical value and a unit of measurement. This allows arithmetic operations between types with different units as well as conversions from and to different units. In this case kph to mph:

```
print(response.value.to("mph"))
```

Creating a custom command

Given that manufacturers may add proprietary PIDs it is important to know how to create new commands for the python OBD module.

In [ ]:
```
cmd = obd.OBDCommand(
                "HYBRID_BATTERY_REMAINING",           # name
                "Hybrid battery pack remaining life", # description
                b"015B",                              # OBD PID 01 (mode) 5B
                3,                                    # number of return byt
                obd.decoders.percent,                 # decoding function
                obd.protocols.ECU.ENGINE,             # (optional) ECU filte
                True)                                 # (optional) allow a '

response = connection.query(cmd) # send the command, and parse the response

print(response.value)

# If you had been connected to a Toyota Auris Hybrid vehicle this would hav
# Unfortunately for the live demo we are restricted to the ELM327 Emulator
```

Now let's create a custom command for something we know the emulator will recognize

In this example lets create our own version of the SPEED command

Remember, you can see the standard definition of the SPEED PID here:
https://en.wikipedia.org/wiki/OBD-II_PIDs (https://en.wikipedia.org/wiki/OBD-II_PIDs)

```
mode: 01
pid: 0D

Bytes returned 3! here's why:
    mode: 01 (byte 1)
    pid: 0D (byte 2)
    return value (byte 3) speed can be [0..255]
```

Let's postpone a discussion of decoders for later

In [ ]:
```
cmd = obd.OBDCommand(
                "mySPEED",                  # name
                "myVehicle Speed",          # description
                b"010D",                    # OBD PID 01 (mode) 5B (sensor)
                3,                          # number of return bytes to expec
                obd.decoders.uas(0x09),     # decoding function
                obd.protocols.ECU.ENGINE,   # (optional) ECU filter
                True)                       # (optional) allow a "01" to be c

response = connection.query(cmd) # send the command, and parse the response

print(response.value)
```

Continuing we create a custom decoder for our custom command

Here are some usefull links when trying to understand command decoding
https://github.com/brendan-w/python-OBD/blob/master/obd/decoders.py
(https://github.com/brendan-w/python-OBD/blob/master/obd/decoders.py)
https://github.com/brendan-w/python-OBD/blob/master/obd/UnitsAndScaling.py
(https://github.com/brendan-w/python-OBD/blob/master/obd/UnitsAndScaling.py)

1) Why does the author use functools.partial?

Assuming:

```
def sum2(x, y):
    return x + y
```

By calling functools.partial(sum2, 4) you create a new function (a callable, to be precise) that behaves like sum2, but has one positional argument less. That missing argument is always substituted by 4, so that partial(sum2, 4)(2) == sum2(4, 2)

So by using functools.partial, the author can create a function reference which may later be called with the message at decode time.

2) Messages from the OBD adapter are represented as an array of bytes

Remember:

```
mode: 01
pid: 0D

Bytes returned 3! here's why:
    mode: 01 (byte 1)
    pid: 0D (byte 2)
    return value (byte 3) speed can be [0..255]
```

3) Encase the returned value inside a UnitsAndScaling object

```
UAS(signed, scale, unit)(value)
```

In [ ]:

```
import functools
from obd.UnitsAndScaling import Unit, UAS_IDS, UAS

def speed_decoder():
    """ get the corresponding decoder for this UAS ID """
    return functools.partial(decode_speed)

def decode_speed(messages):
    d = messages[0].data[2:]  # chop off mode and PID bytes
    return UAS(False, 1, Unit.kph)(d)

cmd = obd.OBDCommand("mySPEED",           # name
                "myVehicle Speed",        # description
                b"010D",                  # OBD PID 01 (group) 0D (sensor)
                3,                        # number of return bytes to expec
                speed_decoder(),          # decoding function
                obd.protocols.ECU.ENGINE, # (optional) ECU filter
                True)                     # (optional) allow a "01" to be c

response = connection.query(cmd) # send the command, and parse the response

print(response.value)
```

Example of a live session (./assets/2019_Ford_Ranger.png)

In [ ]: