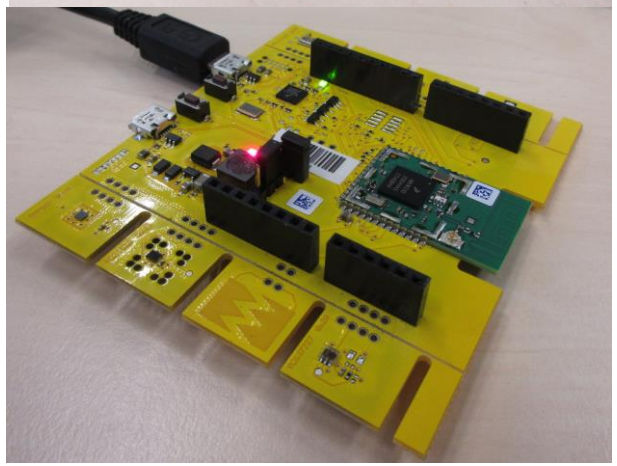
Programmieren lernen mit dem Internet der Dinge

Girls Day 2016

Das Internet der Dinge

- Sensorknoten für alle Gelegenheiten
- Observieren und Verändern der Umwelt
  - Messungen
  - Aktivierung von Geräten
  - drinnen & draussen
- Kommunikation zur Bewältigung komplexer Aufgaben
- z.B. in Haushaltsgeräten, Ampeln, Verkehrsmitteln, Infrastruktur und Natur



Was ist Programmieren?

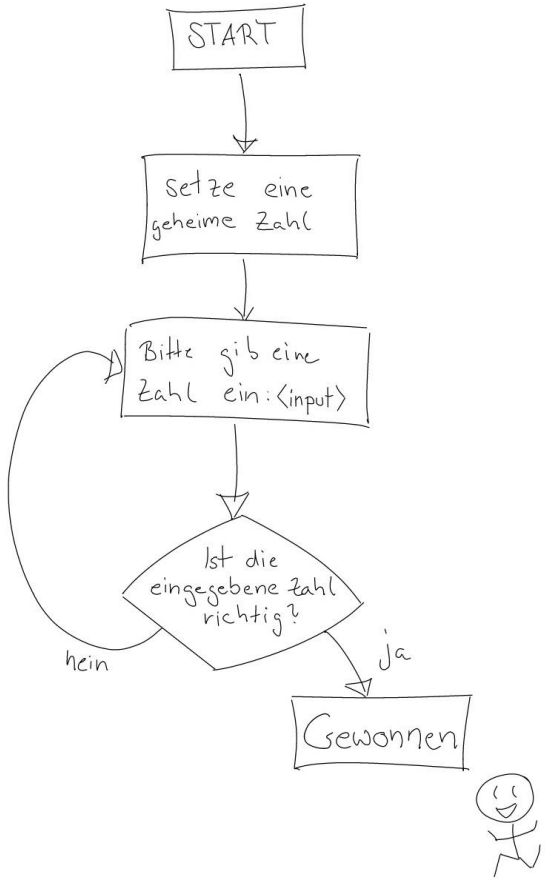
- den Computer eine Aufgabe erledigen lassen
- dem Computer sagen wie er die Aufgabe zu lösen hat
- Programmiersprache als Ausdrucksmittel
  - wiederholende, abstrakte Konzepte, die bestimmte Dinge ausdrücken
  - Python 3

coapserver.py | COAPclient.py | coapclienttest.py | testwrapperclient.py | helloworld.py

```
 7
 8  @asyncio.coroutine
 9  def __request(uri, myMessage, method):
10      logging.debug("started request coroutine")
11      protocol = yield from Context.create_client_context()
12      request = Message(code=method)
13      if (myMessage != ""):
14          request = Message(code=method, payload=myMessage.encode("utf8"))
15      request.set_request_uri(uri)
16
17      try:
18          response = yield from protocol.request(request).response
19      except Exception as e:
20          print('Failed to fetch resource:')
21          print(e)
22      else:
23          print('Result: %s\n%r'%(response.code, response.payload))
24      return response.payload
25
26
27  #"coap://localhost/hello"
28  def get(uri):
29      logging.debug("started GET method with uri: "+uri)
30      response=asyncio.get_event_loop().run_until_complete(__request(uri, "" , GET))
31      return response
```

Spiel: Zahlenraten

- Funktionsweise
  - Computer denkt sich Zahl zwischen 1 und 100 aus
  - Nutzer rät eine Zahl
  - Computer sagt ob Zahl stimmt, *oder*
  - Computer sagt ob Zahl höher oder niedriger ist
- Was braucht man dafür?
  - Informationen merken
  - Informationen anzeigen
  - Nutzereingabe einlesen
  - Testen ob die eingegebene Zahl stimmt
    - und das *immer wieder*



Variablen & Zuweisungen

- Variablen sind Speicher für Informationen
- Kann man später wieder auslesen
- sind nur vorhanden solange das Programm läuft
- Zuweisungen mit Gleichheitszeichen:
  - <Variable>=<Information>
- Variablen sind vom "Typ" ihrer Information
  - `eineZahl` hat den Typ "int"
  - `wort` hat den Typ "String"
  - sehr viele verschiedene Typen: `boolean`, `float`, ...

```
wort="hallo"
variable01="blub"
eineZahl=23
eineAndereZahl=0.1
variable02=wort
```

Funktionen

- Funktionen "tuen" irgendetwas für den Programmierer
- Funktionen werden
  - genutzt um wiederholende Arbeitsabläufe zusammenzufassen
  - können auch selbst geschrieben werden
- Parameter werden Funktionen übergeben
  - mit diesen wird "irgendetwas" gemacht

```
funktion(),
funktion(parameter),
funktion(parameter1, parameter2), usw.
variable.funktion(...)
```

- Funktionen haben häufig ein Ergebnis, das als Rückgabewert zurückgegeben wird
  - diesen kann man in einer Variablen speichern

```
variable01="blub"
eineZahl=23
eineAndereZahl=0.1
variable02=wort

print("hallo")    # gibt "hallo" auf der Kommandozeile aus
print(wort)       # gibt auch "hallo" aus
print(str(eineZahl))       # gibt "23" auf der Kommandozeile aus
print(type(eineAndereZahl)) # zeigt den Typ der Variable "eineAndereZahl"

userinput=input("Some input please:")
    # gibt "Some input please:" aus,
    # lässt den Nutzer schreiben bis <enter> und
    # gibt das was der Nutzer geschrieben hat zurück -> nach userinput

wort.upper()
    #wandelt einen Parameter vom Typ String in Großbuchstaben um
```

Schritt 01 : Aus- & Eingabe

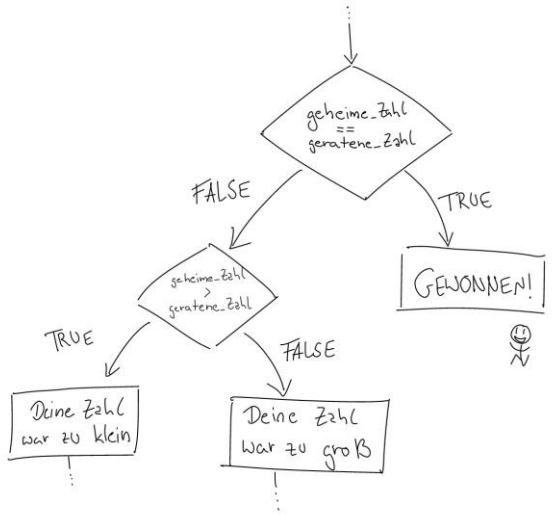
- Schreibe ein Programm, welches "Hello World" auf der Konsole ausgibt
  a) Nutze hierfür die "print"-Funktion
  b) Speichere eine Zeichenkette in einer Variablen und gib diese dann in der Konsole aus.
  c) lies einen String von der Konsole ein und gib ihn in Großbuchstaben wieder aus

```
print("hello world")
blub="hello world again"
print(blub)

#andere funktionen
blub.upper()    # gibt Großbuchstabenversion der
                # Zeichenkette in "blub" zurück

input(blub)     # gibt die Zeichenkette in "blub" aus,
                # und wartet auf Nutzereingabe
```

If-then-else

- Kontrollstruktur:
  - Wenn a dann b,
    (ansonsten c)
- Die Bedingung muss klar mit TRUE oder FALSE beantwortet werden können
  - Datentyp: boolean
  - Vergleichsoperatoren

  ==, !=, <, <=, >, >= , etc
  - Bedeutung ist typabhängig
- Einrückung für Zusammenhang




```
import random

a=1
b=int(random.randint(1,10))

#   Die Bedingung steht in Klammern
if (a == b ):
    print("a und b sind beide: "+str(a))
else:
    print("die Zahlen sind nicht gleich. b war:"+str(b))

#andere Funktionen
random.randint(1,10)    # generiert Zufallszahl zwischen 1 und 10
str(a)                  # macht aus "a" eine Zeichenkette. Nur Zeichenketten
                        # können mit "print" ausgegeben werden
```

Schleifen

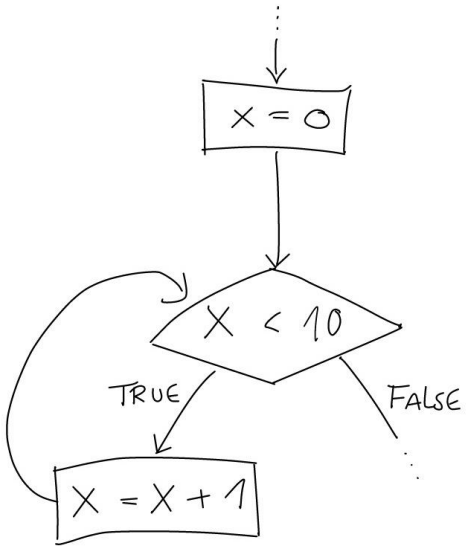
- Wiederholendes Ausführen von Code
- 2 Methoden in Python3:
  - "wiederhole bis Bedingung a auftritt"

```
while <bedingung>:
    <statements>
```

  - "wiederhole für alle <variablen>"

```
for <variable> in <sequence>:
    <statements>
```

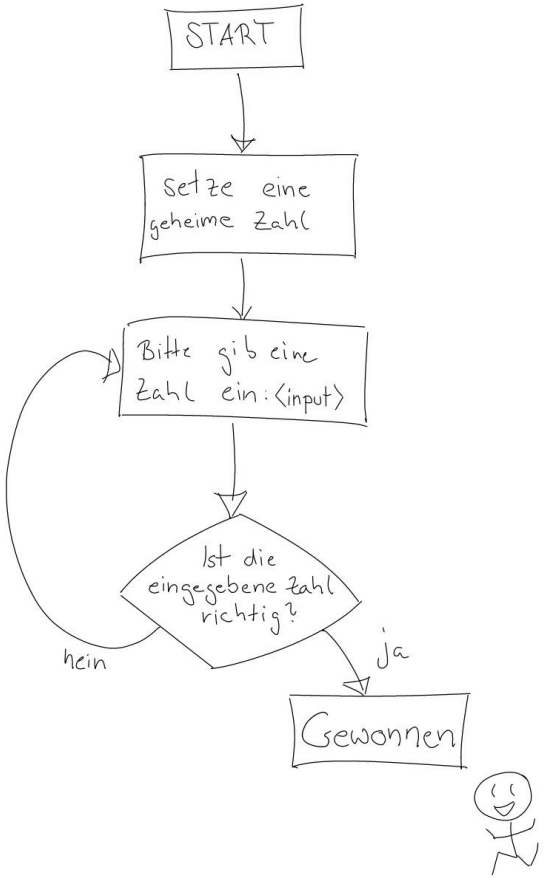
  - wir nutzen erstmal *nur "while"*




```
x=0
while x<10 :
    print("x is "+str(x))
    x=x+1
```

Schritt 02: Zahlenraten

- Programmiere das Zahlenraten-Spiel
  - Nutze Schleifen für wiederholte Abfragen
  - Nutze If-then-else für Vergleiche zwischen der geratenen und der geheimen Zahl
- Erweiterung: Gib aus, ob die geratene Zahl größer oder kleiner der geheimen Zahl ist



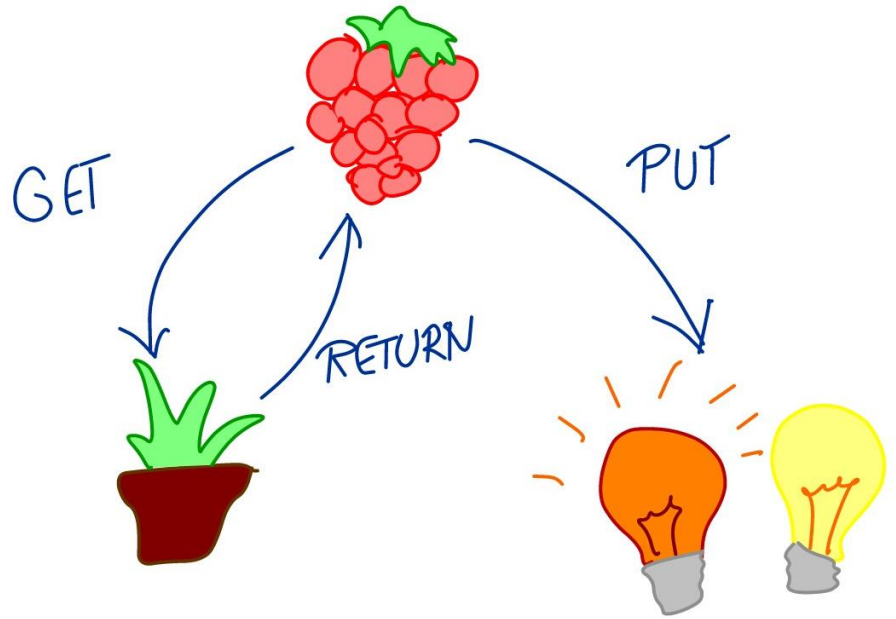
Projekt: Internet der Dinge

- Ziel: Informationen von Sensoren einsammeln, auswerten und beeinflussen
- Wie kommt man an die Sensordaten? Wie kann man Sensoren beeinflussen?
  - Nachrichten verschicken
  - Nachrichten empfangen
- Was kann man aus den Sensordaten berechnen?
  - Schwellwerte
  - Durchschnitt
  - …
- Was braucht man?
  - Kommunikation
  - speichern
  - auswerten

Kommunikation mit COAP

- Information auf dem Knoten spezifierieren
- durch URI (Uniform Resource Identifer) :
  - <protocol>://<Knotenname>/<Ressource>
  - <protocol>://< Knotenadresse>/<Ressource>
  - BSP: coap://plant/humidity
    coap://[fe80::F8E3:4E62:71BA:600A]/r

  Verschiedene Zugriffsarten
  - GET
  - PUT
  - POST
  - DELETE




```
import COAPclient

print("Zuerst testen wir GET")

uri="coap://[fe80::F8E3:4E62:71BA:600A%lowpan0]/temperature"

data=COAPclient.get(uri,"")
print("die temperatur ist: ", data.decode('utf-8'))
```

```
Zuerst testen wir GET
die temperatur ist:  24.66
```

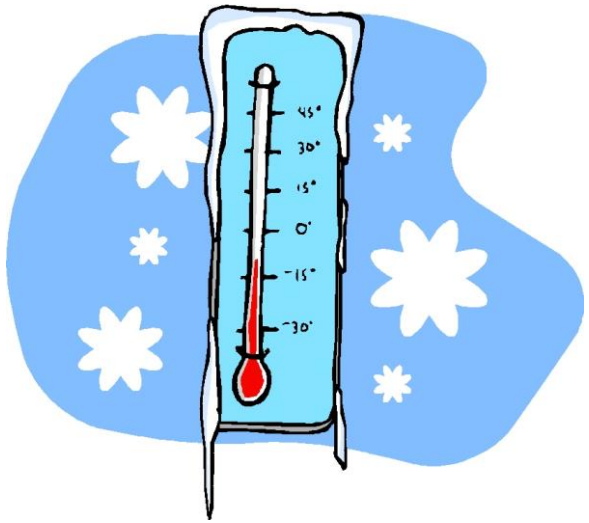
Schritt 03: Internet der Dinge

- Kommuniziere mit den Sensorknoten im Raum
  - Frage die Temperatur und Luftfeuchtigkeit ab
  - Schalte eine LED an und aus
  - Setze die LED auf eine bestimmte Farbe
- Frage diese Daten regelmäßig ab
  - Signalisiere wenn die Temperatur einen bestimmten Schwellwert übersteigt
  - Berechne den Mittelwert der letzten 10 Temperatur-Messungen

```
import COAPclient

uri="coap://[fe80::F8E3:4E62:71BA:600A%lowpan0]/temperature"

# die Kommunikationsmethoden: GET, PUT, POST, DELETE
print("at first we try GET \n")
got_back=COAPclient.get(uri)
print("we got this message back: "+str(got_back))

print("now... GET with message: \n")
got_back=COAPclient.get(uri, "blub")
print("we got this message back: "+str(got_back))

uri2="coap://[fe80::F8E3:4E62:71BA:600A%lowpan0]/led"

print("---\n now we try PUT\n")
COAPclient.put(uri2,"r")
```

Kommunikation mit Sensoren

- GET
  - <uri>/temperature
  - <uri>/humidity



- POST
  - <uri>/led
  - Mögliche Werte:
    - 0
    - 1
    - R
    - G
    - B