Lab 12 –Python / SenseHat (2)

1 import this – the Zen of Python (PEP20)

1.1 The Zen of Python

The Zen of Python is captured in PEP 20: https://www.python.org/dev/peps/pep-0020/

1.2 What's "import this"

The background story: https://www.wefearchange.org/2010/06/import-this-and-zen-of-python.html

So, there is actually a module named "this". The "this" module is found in the usual place (in the Lib directory of your Python installation – Win & Python 3.7: C:\Program Files\Python37\Lib).

The exact code:

```
s = """Gur Mra bs Clguba, ol Gvz Crgref

Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```

If you import it (which here is the same as running the code), you will get the Zen of Python.

Exercise

2 Time is of the Essence

2.1 The EventTimer module with the EvTimer class

Sometimes it is very handy to set up timers to remind us about some event that will happen in the future. There are many ways to do this. Our way is a rather simple one, but it will do for our needs.

See **EventTimer.py** and **TestTimer.py**.

Write a little addition to the EventTimer/EvTimer module/class:

- Add the method Dump() --- test it well

This method (function) prints out the eventlist:

```
Event:      Tag:        Remaining parameters:
<time>      <tag>       <params – if any>
```

Example

```
Event:          - Tag:       - Params:
1539590742.1154   D            0
1539590742.2154   A            1
1539590742.3154   T            2
1539590742.4154   2            3
1539590742.5154   3            4
1539590742.6154   5            5
1539590742.7154                6
1539590742.8154   S            7
1539590742.9154   e            8
1539590743.0154   c            9
1539590743.1154   u            10
1539590743.2154   r            11
1539590743.3154   i            12
1539590743.4154   t            13
1539590743.5154   y            14
1539590743.6154                15
1539590743.7154   a            16
1539590743.8154   n            17
1539590743.9154   d            18
1539590744.0154                19
1539590744.1154   I            20
1539590744.2154   o            21
1539590744.3154   T            22
```

3 Out friend – the silly calculator

A rewrite of the calculator (old files in the attachment).

3.1 The MQTT_Calculator.py program

It shall now be run on the RPI.

After each computation:

- If the accumulator is between 0 and 64:
  - Set **accum** number of pixels on the sensehat to Blue.
- Else:
  - Print accum to using show_message()

3.2 The CalcApp.py file

Our first try at the CalcApppy was actually not very good.

```
def load(num : int):
    publish.single("dat235/calc/oper/load",num, hostname="localhost")
    msg = subscribe.simple("dat235/calc/eval/#", hostname="localhost")
    print("Accum: %s" % (msg.payload))
```

The problem is that the **subscribe.simple()** is likely too late. Also, the call is a blocking call. It may work on a "localhost" with # (which is why that was used), but it is not stable code.

So, we need to rewrite **CalcApp.py** using the **client()** functionality.

- https://www.eclipse.org/paho/clients/python/docs/#client

The outline:

```
import paho.mqtt.client as mqtt     #     Importing paho client
mqttc = mqtt.Client()               #     Create a client object
mqttc.on_connect = on_connect       #     Assign "callback" methods
mqttc.on_message = on_message       #     -- // --
mqttc.connect(host)                 #     Connect to the broker
mqttc.loop()                        #     Check incoming
mqttc.publish(topic,payload,..)     #     Publish from the client




# Callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    sys.stdout.flush()


# Callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic, str(msg.payload))
    # Your code here
```