Appendix 1 - Debugging

"Om debugging är processen att ta bort bugar, då måste programmering vara processen att lägga till dem."

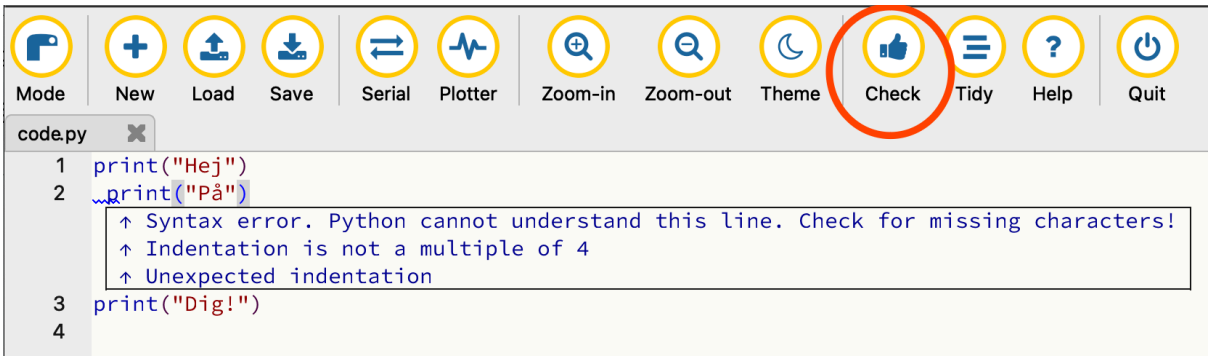
- E. W. Dijkstra

För att tycka om upplevelsen att programmera och för att bli en bra programmerare behöver man kunna hantera fel och felmeddelanden. Detta appendix är menat att ge instruktioner om det. Detta dokument är inte labb eftersom det inte passar någonstans i ordningen, utan är menat att användas vid behov under tiden de ordinarie labbarna görs.

Hur hanterar man ett fel eller felmeddelande? Generellt består processen av tre steg,

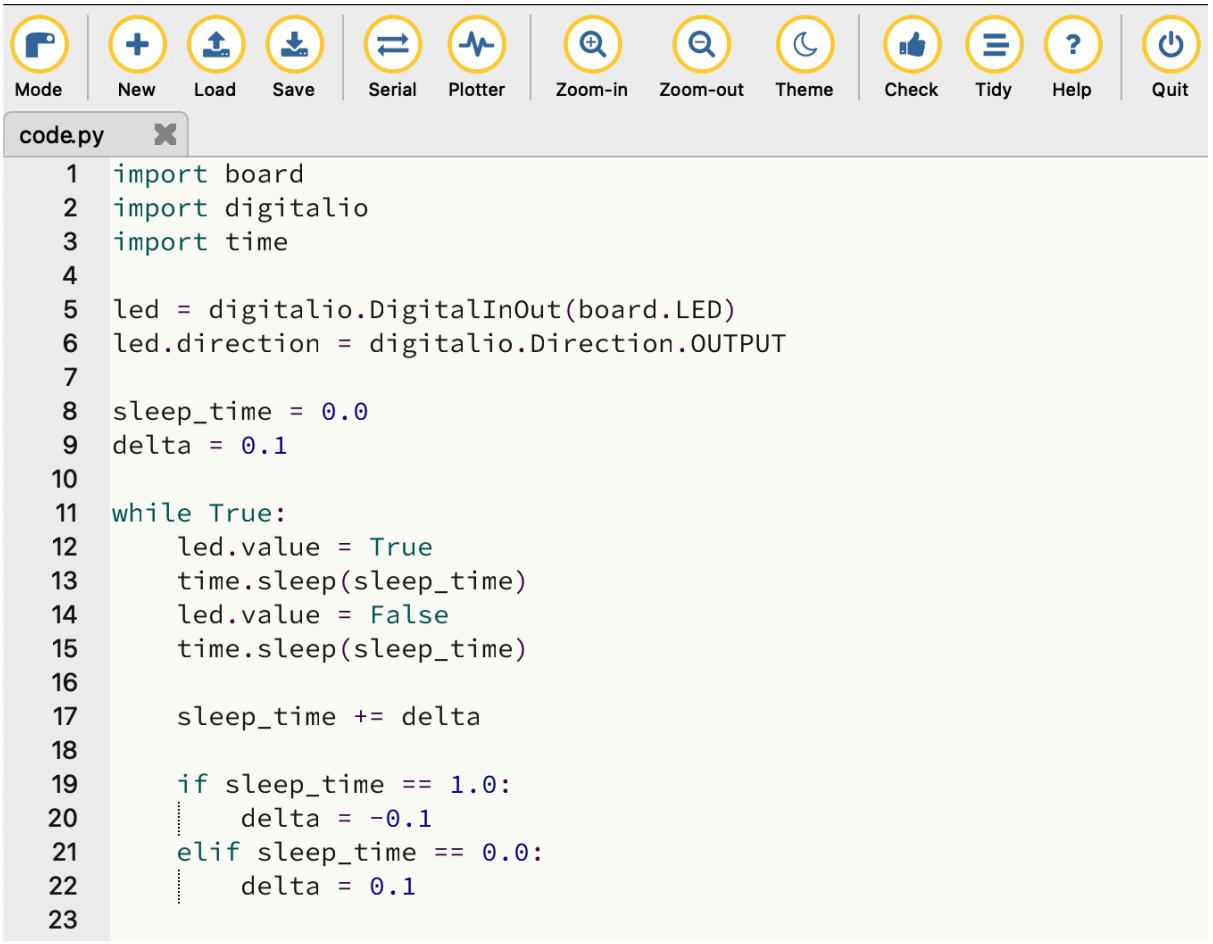
1. Identifiera problemet
2. Bestäm lösningen på problemet
3. Implementera lösningen

Men det är lättare sagt än gjort. Hur identifierar man ett problem? För att få en tydligare bild av vart problemet kan härstamma ifrån kan man använda sig av **Check** funktionen.



Felmeddelandet säger att det finns ett oväntat blanksteg på rad 2. Då Python är känsligt för blanksteg måste alla kodrader som tillhör samma kodblock ha lika många blanksteg innan sig.

Ett bytedyligt svårare exempel är följande




```
1  import board
2  import digitalio
3  import time
4
5  led = digitalio.DigitalInOut(board.LED)
6  led.direction = digitalio.Direction.OUTPUT
7
8  sleep_time = 0.0
9  delta = 0.1
10
11 while True:
12     led.value = True
13     time.sleep(sleep_time)
14     led.value = False
15     time.sleep(sleep_time)
16
17     sleep_time += delta
18
19     if sleep_time == 1.0:
20         delta = -0.1
21     elif sleep_time == 0.0:
22         delta = 0.1
23
```

Denna är betydligt svårare. Koden ger inga felmeddelanden men lysdioden ökar sin puls tid för evigt utan att vända om vid 1 sekund. En bra strategi när programmet inte säger att något är fel är att få det att skriva ut mer data. Det görs med Serial monitorn. Genom att modifiera programmet enligt bilden nedan kan man se vad som händer.

Mode | New | Load | Save | Serial | Plotter | Zoom-in | Zoom-out | Theme | Check | Tidy | Help | Quit

code.py

```
1  import board
2  import digitalio
3  import time
4
5  led = digitalio.DigitalInOut(board.LED)
6  led.direction = digitalio.Direction.OUTPUT
7
8  sleep_time = 0.0
9  delta = 0.1
10
11 while True:
12     led.value = True
13     time.sleep(sleep_time)
14     led.value = False
15     time.sleep(sleep_time)
16
17     sleep_time += delta
18
19     if sleep_time == 1.0:
20         delta = -0.1
21     elif sleep_time == 0.0:
22         delta = 0.1
23
24     print("Sleep time: ", sleep_time, "Delta:", delta)
25
```

CircuitPython REPL

```
Sleep time:  0.7 Delta: 0.1
Sleep time:  0.799999 Delta: 0.1
Sleep time:  0.899999 Delta: 0.1
Sleep time:  0.999999 Delta: 0.1
Sleep time:  1.1 Delta: 0.1
Sleep time:  1.2 Delta: 0.1
Sleep time:  1.3 Delta: 0.1
```

Nu kommer programmet tala om vilket värde *`sleep_time`* och *`delta`* har varje gång det går igenom loopen. Genom att kolla i Serial monitor ser vi att *sleep_time* går från 0.7 till 0.799999 sedan 0.899999 sedan 0.999999 för att slutligen passera 1 utan att någonsin vara 1 så att if satsens kriterium aldrig blir sant. Detta uppstår då datorer har svårt för att representera och jämföra decimaltal, eller så dem kallas inom programmering, flyt tal. Ett bra tips är att undvika flyt tal så långt det går. Till exempel skulle man kunna skriva om koden till att arbeta med hela millisekunder istället för decimala sekunder.

Mode | New | Load | Save | Serial | Plotter | Zoom-in | Zoom-out | Theme | Check | Tidy | Help | Quit

code.py

```
1  import board
2  import digitalio
3  import time
4
5  led = digitalio.DigitalInOut(board.LED)
6  led.direction = digitalio.Direction.OUTPUT
7
8  sleep_time_ms = 0
9  delta_ms = 100
10
11 while True:
12     led.value = True
13     time.sleep(sleep_time_ms / 1000)
14     led.value = False
15     time.sleep(sleep_time_ms / 1000)
16
17     sleep_time_ms += delta_ms
18
19     if sleep_time_ms == 1000:
20         delta_ms = -100
21     elif sleep_time_ms == 0:
22         delta_ms = 100
23
24     print("Sleep time: ", sleep_time_ms, "Delta:", delta_ms)
25
```

CircuitPython REPL

```
Sleep time:  800 Delta: 100
Sleep time:  900 Delta: 100
Sleep time:  1000 Delta: -100
Sleep time:  900 Delta: -100
Sleep time:  800 Delta: -100
Sleep time:  700 Delta: -100
Sleep time:  600 Delta: -100
```

Koden fungerar nu som förväntat.

Från dessa exempel kan vi se en process för problemlösning. Den ser ut ungefär såhär

1. Läs och tolka eventuella felmeddelanden
2. Få programmet att skriva ut mer information
3. Läs dokumentationen
4. Googla problemet

Den fjärde punkten är inte något vi har pratat om än. Anledningen till att CircuitPython används i dessa labbar är att det är många andra som har använt CircuitPython tidigare. Dessa människor har antagligen redan haft alla problem vi kommer att stöta på. Genom att klistra in ett felmeddelande i Google kommer ett svar nästan säkert dyka upp. Att söka på engelska kan också vara en god ide då majoriteten av tillgängligt material på området är just på engelska.