**1) Given the following program:**

live\_loop :foo do

play 60

sleep 1

end

**Now press the Run button. You hear a basic beep every beat. However, don’t press Stop just yet. Change the 60 to 65 and press Run again. What happens?**

1. No effect on execution. This is live coding.
2. It forks the main thread and creates two new ones. One will play 60 and the other 65. This is live coding.
3. It changed automatically without missing a beat. This is live coding.
4. It raises a RunTime error.

**2) Given the following Sonic Pi code:**

live\_loop :foo do

          play 50

          sleep 1

end

sample :drum\_cymbal\_open

**What happens if you run it?**

1. The loop will repeat 50 times and then play the sample.
2. The loop and the sample are executed simultaneously.
3. All the code after the loop is not executed.
4. The drum cymbal open sample is played while the the loop is "asleep".

**3) Given the following Sonic Pi code:**

live\_loop :foo do

  sample :ambi\_choir

  sleep 0.5

end

in\_thread do

  sample :ambi\_drone

end

**4) What happens if you run it?**

1. It is not possible to execute both a thread and a live loop concurrently (runtime error).
2. Both the live\_loop and the thread sample are played infinitely.
3. The live\_loop sample is played infinitely, while the thread sample is played only once.
4. Only the live\_loop will be executed.

**5) What are the Set and Get functions of the Sonic Pi for?**

live\_loop :setter do

set :foo, rrand(70, 130)

sleep 1

end

live\_loop :getter do

puts get[:foo]

sleep 0.5

end

1. They allow threads to access a shared resource in a thread-safe way, but mutual exclusion is not guaranteed.
2. They are used to produce non-deterministic program behaviour.
3. They are used to manipulate objects whose scope is restricted to a single thread or function.
4. They allow threads to access a shared resource in a thread-safe way, with guaranteed mutual exclusion.

**6) Given the following Sonic Pi code:**

live\_loop :foo do

    use\_synth :prophet

    play 20

    sleep 8

    cue :f

end

sleep 0.3

live\_loop :bar do

    sync :f

    sample :bd\_haus

    sleep 0.5

end

**Are the two live loops synchronized?**

1. The code is incorrect and will not be executed.
2. The two live loops are not synchronized because the get and set methods are not used.
3. The two live loops are out of phase due to the 0.3 sleep between the two live loops.
4. The two live loops are synchronized because the ith iteration of the live loop: bar is synchronized with the ith iteration of the loop: foo via sync, which waits for the cue :f event.