Melodic Sequences Activity

In this lesson, you will be using **for** and **while** loops to create melodies! As with the beat exercise, try to do everything from memory as much as possible, and then click through to the help and solutions for extra guidance.

Setup

The first few steps are more-or-less the same as in the beat-making activity, except that we will be using a different default soundfont.

1. As before, start by writing the code to import everything from the scamp library. Then, create a new Session object and assign it to the variable `s`. When creating the Session `s`, include a keyword argument `default_soundfont` and set it to the string "Synths.sf2", so that we are using the soundfont with the synth sounds. Help, Solution
2. As before, add a line of code to ask `s` to print the available instrument sounds. (Remember, you can start typing `s.print` and then hit tab to have it give you suggestions.) Try running the code, so that it prints out the possible sounds. Solution
3. Add a line asking `s` to create a new part, using one of the instrument sounds you just printed, and store this in an appropriately named variable. Help, Solution
4. Ask `synth` to play some notes using the `play_note` function! Put a bunch of `play_note` calls one after the other and try to create an interesting melody. You might also try changing the instrument sound. Help, Solution

For loops

5. We're going to start doing some experiments with for loops, so for now, **comment out** the `play_note` calls you just used to make your melody. The easiest way to do this is to select the lines and press Command-3. With the lines still selected, press Command-3 again. And again. Notice that it toggles the comments on and off. Leave them commented out (with the hashtags there) so that we can play around with some new code and not have to hear the melody every time. You might also comment out `s.print_default_soundfont_presets`. Solution

Looping through a range

6. Write a for/range loop to play a rising sequence of notes. Remember, this looks like `for [variable] in range([start], [stop]):`. In this case, you could name the variable `pitch`, and then pass it as the first argument to `play_note`. You might also play around with adding a *third* argument to range, which makes it skip up by a larger number each time. Help, Solution
7. This is where it gets fun. Try putting *multiple* `play_note` calls, all indented, inside of the for loop. The first might play at `pitch`, but the second could play at `pitch + 4`, say, and the third could play at `pitch - 3`. This will cause the computer to play a rising musical *sequence*. Help
8. Now, inside the for loop, let's have the code branch into two different possibilities, depending on whether the pitch is even or odd. Play one set of notes when the pitch is even, and another when it is odd. Help, More help

Looping through a list

9. Comment out that for loop and start a new one, but this time, make it loop through a list of numbers, playing a note at each of those pitches. Remember, a list is created by putting numbers (or any kind of data, actually) inside of square brackets. Help, More help
10. Now, try the same thing that you did in step 7: put multiple `play_note` statements inside of the for loop. This can let you play the same gesture high and low and all over the place! Help

Exploring!

11. Now it's time to explore! Uncomment all of the code that you have so far, and save a copy of the file under a new name by selecting "File > Save As…". This way you don't have to worry about losing what you've done. Now, play around with all the techniques we've discussed and make an interesting melody! You might also try:

    - Mixing in some single `play_note` calls (or parts of the melody that you made in part 4) in between the for loops.
    - Putting loops inside of other loops!

- Using the loop variable for something other than pitch (maybe duration?)
- Using if statements and the modulo operator in interesting ways. (For instance, you could do something different on every third pitch, or use the if statement to determine which of several different instruments plays the note.)

It's also totally alright to keep it simple, and just try to make something that you like the sound of! Help

12. Wrap the entire thing in a while loop, so that you can listen to it forever. Then add the line `playback_settings.recording_file_path = melodies.wav` right after importing everything from scamp to save a a recording! Help

Help 1a

Importing things

To import something from a library in Python, you use the following syntax:

```
from [library] import [thing we want to use]
```

So, for instance, if we have a library called "pets", and we want to import the Dog class, we might write:

```
from pets import Dog
```

To import *everything* from a library, we can use an asterisk. So to import all sorts of pets from the pets library, we might write:

```
from pets import *
```

What line of code would you use to import everything from the scamp library?

Creating an object

To create an object of type `Dog`, we would use the syntax `Dog()`. Remember that this is like calling a function, but because dog is uppercase, it means that we're creating a new object.

To **assign** this new dog to the **variable** my_dog, we would write:

```
my_dog = Dog()
```

Finally, if we wanted to use the **keyword argument** "ears" to specify that the ears should be floppy, we might write:

```
my_dog = Dog(ears="floppy")
```

Notice that, since it's surrounded by quotes, the word "floppy" is a **string**.

So how would you write a new line of code to create a new `Session` object, setting the `default_soundfont` keyword argument to "Synths.sf2", and assigning it to the variable `s`?

Back to instructions

Help 1b

You can import the scamp libraries with the line:

```
from scamp import *
```

The * means everything. One of the things we are importing is the `Session` class. To create a new Session, you can use the line:

```
s = Session()
```

To make it use the "Synths.sf2" soundfont, which has all the sounds we want to be using, add a keyword argument:

```
s = Session(default_soundfont="Synths.sf2")
```

So at this point, your code would read:

```
from scamp import *
s = Session(default_soundfont="Synths.sf2")
```

Back to instructions

Help 2

Simply add the line:

```
s.print_default_soundfont_presets()
```

At this point, your code should read:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")

s.print_default_soundfont_presets()
```

When you run it, you should get a printout like:

```
PRESETS FOR synths
   Preset[000:000] LD-AcidSQneutral 2 bag(s) from #0
   Preset[000:001] LD-DanceTrance 2 bag(s) from #2
   Preset[000:002] CandyBee 2 bag(s) from #4
   Preset[000:003] SCP-Beeper 2 bag(s) from #6
   Preset[000:004] AnalogSaw1 2 bag(s) from #8
   Preset[000:005] PlasticStrings 2 bag(s) from #10
   Preset[000:006] BS-DirtySub 2 bag(s) from #12
   Preset[000:007] hypersawwave 2 bag(s) from #14
   Preset[000:008] PD-Sinewave 2 bag(s) from #16
   Preset[000:009] LD-PolySpecialMono 2 bag(s) from #18
   Preset[000:010] PulseWobblerA 2 bag(s) from #20
   Preset[000:011] SuperSawA 2 bag(s) from #22
   Preset[000:012] DY-Synthe 2 bag(s) from #24
   Preset[000:013] SupSawA 2 bag(s) from #26
   Preset EOP
```

Back to instructions

Help 3a

Remember in our hello world example, there was the line:

```
piano = s.new_part("piano")
```

Remember that `"piano"` with quotes is a **string** (text data) representing the sound that you want to use, and `piano` without quotes is the name of the variable in which we store the instrument.

Can you add some modified version of this line to your code?

Back to instructions

Help 3b

To add a new synth, use a line like:

```
synth = s.new_part("CandyBee")
```

Unlike the beat activity, in this activity we will probably only really need one instrument, though you could use more if you want to. At this point, your code should look like:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")

s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")
```

But use whatever instrument you want to experiment with! You can also change it later.

Back to instructions

Help 4a

In my case since the instrument is stored in the variable **`synth`**, I can play a note by writing:

```
synth.play_note(60, 0.7, 1.0)
```

Remember that the **arguments** go in the order pitch, volume, length. So you could make it louder by changing the 0.7 to an 0.9, and you could make it shorter by changing the 1.0 to an 0.5.

Can you add several of these lines to your code, changing the pitches, volumes, and lengths to get an interesting melody?

Back to instructions

Help 4b

For example, add some lines like:

```
synth.play_note(70, 0.7, 1.5)
synth.play_note(69, 0.8, 0.5)
synth.play_note(70, 0.9, 0.5)
synth.play_note(67, 1.0, 1.0)
synth.play_note(66, 1.0, 1.0)
synth.play_note(62, 0.9, 0.5)
synth.play_note(63, 0.8, 0.5)
synth.play_note(62, 0.7, 2.5)
```

So the full code at this point might look like:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

synth.play_note(70, 0.7, 1.5)
synth.play_note(69, 0.8, 0.5)
synth.play_note(70, 0.9, 0.5)
synth.play_note(67, 1.0, 1.0)
synth.play_note(66, 1.0, 1.0)
synth.play_note(62, 0.9, 0.5)
synth.play_note(63, 0.8, 0.5)
synth.play_note(62, 0.7, 2.5)
```

Back to instructions

Help 5

After commenting out the lines, your code should look something like this:

```python
from scamp import *

s = Session(default_soundfont="Synths.sf2")
# s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)
```

Back to instructions

Help 6a

Remember that a for/range loop allows you to run the same code several times for different values of a variable. For example, we could print the numbers from 30 to 80 with the following code:

```python
for x in range(30, 80):
    print(x)
```

It doesn't matter what we call the variable. It could even have been called `ice_cream_is_delicious`:

```python
for ice_cream_is_delicious in range(30, 80):
    print(ice_cream_is_delicious)
```

In our case, it makes sense to call the variable something like `pitch`, and instead of printing the value of `pitch`, we want to use it in a `play_note` call. How might that look?

Back to instructions

Help 6b

To loop through and play a chromatic scale from 55 to 66, we might write:

```python
for pitch in range(55, 67):
    synth.play_note(pitch, 0.8, 0.25)
```

Note that this will not play pitch 67! It always includes the first number but not the last number.

If you want to skip up by something other than one every time, you can add a third argument. For instance, this code plays a whole tone scale from 50 to 70:

```python
for pitch in range(50, 71, 2):
    synth.play_note(pitch, 0.8, 0.25)
```

Of course, you can change the volume and duration arguments to whatever you would like.

At this point, the full code might look something like:

```python
from scamp import *

s = Session(default_soundfont="Synths.sf2")
# s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)

for pitch in range(55, 67):
    synth.play_note(pitch, 0.8, 0.25)
```

Back to instructions

Help 7

Just like you can play a note using `pitch`, you can play one using any variation of `pitch`, adding or subtracting something, even multiplying or dividing. Remember that you need to indent all of the lines that you want to loop:

```
for pitch in range(55, 67):
    synth.play_note(pitch, 1, 0.25)
    synth.play_note(pitch + 4, 1, 0.25)
    synth.play_note(pitch - 3, 1, 0.25)
```

The for loop creates a temporary variable named `pitch` which is only alive inside of the indented block. What happens if you forget to indent the extra lines?

```
for pitch in range(55, 67):
    synth.play_note(pitch, 1, 0.25)
synth.play_note(pitch + 4, 1, 0.25)
synth.play_note(pitch - 3, 1, 0.25)
```

Try playing around with how many `play_note` calls you put inside the loop. Try changing their volumes and their lengths. You might also see what happens if one of the `play_note` calls does not use the pitch variable, like this:

```
for pitch in range(55, 67):
    synth.play_note(70, 1, 0.25)
    synth.play_note(pitch, 1, 0.25)
    synth.play_note(pitch + 4, 1, 0.25)
```

At this point, you code might look something like this:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
```

```python
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)

for pitch in range(55, 67):
    synth.play_note(pitch, 0.8, 0.25)
    synth.play_note(pitch + 4, 0.8, 0.25)
    synth.play_note(pitch - 3, 0.8, 0.25)
```

Back to instructions

Help 8a

Remember that when you want the program to branch into one of two possibilities depending on a condition, you can use an `if` statement. For instance, to play one thing when the pitch variable is greater that 60, and another when it is less (or equal), you could write:

```
if pitch > 60:
    # play one thing
else:
    # play something else
```

You don't need the else clause, by the way; you could also write:

```
if pitch > 60:
    # play one thing
```

…which will just do nothing if the pitch is 60 or lower.

To test whether or not the pitch is even or odd, we will need the `%` (modulo) operator, which gives the remainder when two numbers are divided. For instance, `7 % 3` is 1, and `19 % 5` is 4. To test if a number is even, we can simply check whether it has a remainder of zero when we divide by 2 (`x % 2 == 0`).

Can you use this to make it play one thing for even pitches and another for odd?

Back to instructions

Help 8b

To do one thing when `pitch` is even and another when it's odd, write:

```
if pitch % 2 == 0:
    # do one thing
else:
    # do another
```

Remember, because the single equals sign, `=`, is used to set the value of a variable (it's called the "assignment operator"), we need to use the double equals , `==`, to test if values are equal.

This will play one gesture when `pitch` is even and another when it is odd:

```
for pitch in range(55, 67):
    if pitch % 2 == 0:
        synth.play_note(pitch, 0.8, 0.25)
        synth.play_note(pitch + 4, 0.8, 0.25)
        synth.play_note(pitch - 3, 0.8, 0.25)
    else:
        synth.play_note(pitch, 0.8, 0.125)
        synth.play_note(pitch + 2, 0.8, 0.125)
```

At this point, your script might look something like:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
# s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)
```

```python
for pitch in range(55, 67):
    if pitch % 2 == 0:
        synth.play_note(pitch, 0.8, 0.25)
        synth.play_note(pitch + 4, 0.8, 0.25)
        synth.play_note(pitch - 3, 0.8, 0.25)
    else:
        synth.play_note(pitch, 0.8, 0.125)
        synth.play_note(pitch + 2, 0.8, 0.125)
```

Back to instructions

Help 9a

A list of numbers is created with square brackets, like this: `[56, 67, 62, 89]`

To use a list like this in a for loop, you simply write `for [variable name] in [56, 67, 62, 89]:`

For example, if we wrote:

```python
for x in [56, 67, 62, 89]:
    print(x)
```

It would print out:

```
56
67
62
89
```

As before, it doesn't matter what you call the variable. You'll get the same result with:

```python
for ice_cream_is_delicious in [56, 67, 62, 89]:
    print(ice_cream_is_delicious)
```

In our case, it probably makes the most sense to call the variable `pitch` again. How might you loop through a list of pitches, and play a note for each one?

Back to instructions

Help 9b

To play a sequence of notes at different pitches, you could write something like:

```
for pitch in [56, 70, 64, 72, 80]:
    synth.play_note(pitch, 1, 0.25)
```

After commenting out the previous for/range loop (using command-3) and adding this new loop, your code should look something like this :

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)

# for pitch in range(55, 67):
#     if pitch % 2 == 0:
#         synth.play_note(pitch, 0.8, 0.25)
#         synth.play_note(pitch + 4, 0.8, 0.25)
#         synth.play_note(pitch - 3, 0.8, 0.25)
#     else:
#         synth.play_note(pitch, 0.8, 0.125)
#         synth.play_note(pitch + 2, 0.8, 0.125)

for pitch in [56, 70, 64, 72, 80]:
    synth.play_note(pitch, 1, 0.25)
```

Back to instructions

Help 10

Just like you did before with the for/range loop, all you have to do is indent multiple play_note lines, and have each one add or subtract something to the `pitch` variable before passing it to the `play_note` function. You can also use different durations:

```
for pitch in [56, 70, 64, 72, 80]:
    synth.play_note(pitch, 1, 0.125)
    synth.play_note(pitch + 4, 1, 0.625)
    synth.play_note(pitch - 3, 1, 0.25)
```

What's fun about this is that you can use it to play the same gesture up, down, and all over the place, just by changing the numbers inside the list!

At this point, your code might look something like this:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

# synth.play_note(70, 0.7, 1.5)
# synth.play_note(69, 0.8, 0.5)
# synth.play_note(70, 0.9, 0.5)
# synth.play_note(67, 1.0, 1.0)
# synth.play_note(66, 1.0, 1.0)
# synth.play_note(62, 0.9, 0.5)
# synth.play_note(63, 0.8, 0.5)
# synth.play_note(62, 0.7, 2.5)

# for pitch in range(55, 67):
#     if pitch % 2 == 0:
#         synth.play_note(pitch, 0.8, 0.25)
#         synth.play_note(pitch + 4, 0.8, 0.25)
#         synth.play_note(pitch - 3, 0.8, 0.25)
#     else:
#         synth.play_note(pitch, 0.8, 0.125)
#          synth.play_note(pitch + 2, 0.8, 0.125)

for pitch in [56, 70, 64, 72, 80]:
    synth.play_note(pitch, 1, 0.125)
    synth.play_note(pitch + 4, 1, 0.625)
    synth.play_note(pitch - 3, 1, 0.25)
```

Back to instructions

Help 11

A simple approach would be to mix together for loops and `play_note` calls to get something like this:

```
from scamp import *

s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

for pitch in [56, 70, 64]:
    synth.play_note(pitch, 1, 0.25)
    synth.play_note(pitch + 4, 1, 0.25)

synth.play_note(50, 1, 2)

for pitch in range(52, 62):
    if pitch % 2 == 0:
        synth.play_note(pitch, 0.8, 0.125)
        synth.play_note(pitch + 4, 0.8, 0.25)
        synth.play_note(pitch - 3, 0.8, 0.125)
    else:
        synth.play_note(pitch, 0.8, 0.125)
        synth.play_note(pitch + 2, 0.8, 0.125)

synth.play_note(51, 1, 0.125)
synth.play_note(53, 1, 0.125)
synth.play_note(54, 1, 0.75)
```

Notice that all of the `play_note` calls within the for loops are indented, and all of the `play_note` calls outside of the for loops are not. Also, notice that within the for loops you can use the `pitch` variable, but outside of the for loops you cannot.

Play around and see what you can come up with! You could make it slow or fast, or mix up fast sweeps like this…

```
for pitch in range(62, 80):
    synth.play_note(pitch, 1, 0.05)
```

… with long notes like this:

```
synth.play_note(58, 0.6, 3)
```

You might use some calls to the `wait` function to add rests, or you might try add `blocking=False` to some of the `play_note` calls to make things overlap.

Finally, we haven't really talked about this, but you can also try playing chords by using the `play_chord` function. It works like this:

```
synth.play_chord([60, 67, 70, 78, 86], 1, 3)
```

Instead of a single number as the pitch, it takes a list of numbers so that it can play several pitches at once.

Back to instructions

Help 12

Remember that to make several lines of code happen over and over forever, all you need to do is place `while True:` before them, and then indent them:

```python
while True:
    # put some code here that you want
    # to repeat forever
```

In our case, we would take *all* of that code you just wrote — for loops and play note calls and all — and indent the whole thing! In my example, it turned out something like this:

```python
from scamp import *

playback_settings.recording_file_path = "melodies.wav"
s = Session(default_soundfont="Synths.sf2")
s.print_default_soundfont_presets()

synth = s.new_part("CandyBee")

while True:
    for pitch in [56, 70, 64]:
        synth.play_note(pitch, 1, 0.25)
        synth.play_note(pitch + 4, 1, 0.25)

    synth.play_note(50, 1, 2)

    for pitch in range(52, 62):
        if pitch % 2 == 0:
            synth.play_note(pitch, 0.8, 0.125)
            synth.play_note(pitch + 4, 0.8, 0.25)
            synth.play_note(pitch - 3, 0.8, 0.125)
        else:
            synth.play_note(pitch, 0.8, 0.125)
            synth.play_note(pitch + 2, 0.8, 0.125)

    synth.play_note(51, 1, 0.125)
    synth.play_note(53, 1, 0.125)
    synth.play_note(54, 1, 0.75)
```

(Notice the line `playback_settings.recording_file_path = "melodies.wav"` which lets you save the output to a WAV file.)

Excited to see what you come up with!

Back to instructions