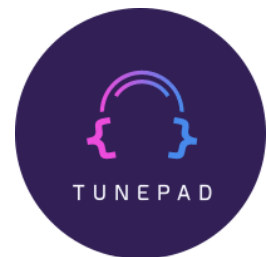


Learning Python through coding music

September, 2022

```
6 G = [55, 59, 62]
7 E7 = [52, 56, 59, 62]
8
9 listHal = [C, Am, C, Am, F, G, C, G, C, F, G, Am, F, G, E7, Am]
10 durHal = [6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 6, 6, 6, 6, 6]
11 listChorus = [F, F, F, F, Am, Am, Am, Am, F, F, F, F, C, C, G, G, C, C]
12 for chord, duration in zip(listHal, durHal):
13     for dur in range(0, duration):
```

READER



Krijn Hoogendorp

TunePad was created by the [TIDAL Lab](#) at Northwestern University.
TunePad owns and retains all rights to the TunePad code, design, and functionality.
All user-generated content on TunePad is licensed under the Creative Commons Attribution-ShareAlike 2.0 license.

EarSketch is a product from Georgia Tech.
All rights to EarSketch code, design, and functionality is owned and retained by Georgia Tech Research Corporation.
The EarSketch curriculum and teaching materials are licensed under [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International \(CC BY-NC-SA 4.0\)](#).

References are made to the excellent book *Introduction to Digital Music with Python Programming* By Michael S. Horn, [Melanie West](#), [Cameron Roberts](#)

"Python" is a registered trademark of the Python Software Foundation (PSF). The Python logos are use trademarks of the PSF as well.

This reader was created by Krijn Hoogendorp and is licensed under [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International \(CC BY-NC-SA 4.0\)](#). Contact if you like to redistribute (parts of-) the reader is appreciated.

Learning Python through coding music.....	5
Lesson 1: TunePad	6
<i>Python intermezzo 1</i>	<i>10</i>
<i>Book reference</i>	<i>14</i>
<i>Assignment 1</i>	<i>14</i>
Lesson 2: Tempo and rhythm	16
<i>Python intermezzo 2</i>	<i>19</i>
<i>Book reference</i>	<i>20</i>
<i>Assignment 2</i>	<i>20</i>
Lesson 3: TunePad melody	21
<i>Python intermezzo 3</i>	<i>23</i>
<i>Book reference</i>	<i>26</i>
<i>Assignment 3</i>	<i>26</i>
Lesson 4: Chords	29
<i>Python intermezzo 4</i>	<i>32</i>
<i>Book reference</i>	<i>33</i>
<i>Assignment 4</i>	<i>33</i>
Lesson 5: Scales, keys and melody.....	34
<i>Python intermezzo 5</i>	<i>35</i>
<i>Book reference</i>	<i>37</i>
<i>Assignment 5</i>	<i>37</i>
Lesson 6: Diatonic chords and chord progression	38
<i>Python intermezzo 6</i>	<i>39</i>
<i>Book reference</i>	<i>41</i>
<i>Assignment 6</i>	<i>41</i>
Final Exercise week 1	42
les 7: EarSketch intro	44
<i>Python intermezzo 7</i>	<i>47</i>
<i>Book reference</i>	<i>51</i>
<i>Assignment lesson 7.....</i>	<i>51</i>
lesson 8: EarSketch remix.....	51
<i>Python intermezzo 8</i>	<i>54</i>
<i>Book reference</i>	<i>57</i>
<i>Assignment 8</i>	<i>58</i>
Final assignment week 2	59



Learning Python through coding music

Welcome to the course **Learning Python through coding music**.

We hope you will enjoy the course. Programming is increasingly getting an essential asset in many professional fields, but even more important, coding is also fun.

In this course you will work with basic programming concepts such as *variables*, *if-selection*, *loops*, *functions* and much more. No previous programming knowledge is needed, but even if you have extensive experience with coding you might still learn a lot if you challenge yourself to create complex musical tunes.

We do not expect you to have a music background. The creators of this course are IT specialist and certainly not musicians. However, some music theory will be addressed in a *quick-and-dirty* manner. No worries if you do not understand. You can rely on your ears.

Every day there will be two one-hour lectures (at 9:00 and at 13:30 hours) followed by practical exercises. In week two you will create your own composition with Python.

In the first week we use the online programming environment TunePad and in week two we move to EarSketch (another online environment).

TunePad is well suited for the basics of Python. EarSketch is designed to use Python on music samples and provides the possibility for the creation of your own composition (which you will present at the end of week 2).

The practical exercises are explained in this document and in the book *Introduction to Digital Music with Python Programming* (Horn, West, Roberts, 2022). You do not need to purchase the book. We will provide a copy.

Music is an important part of the daily lectures, as inspiration, but also to explain the similarities between programming and music. For sure you will think of your own examples of suitable songs. Please share your thoughts with us.

Some EarSketch examples:

<https://youtu.be/JqkWljZ3EW0>

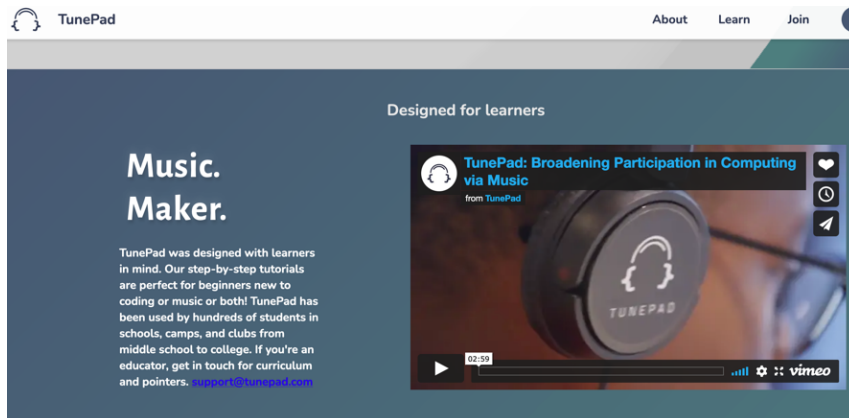
On how to learn music (and the same applies to programming):

<https://youtu.be/3yRMbH36HRE>

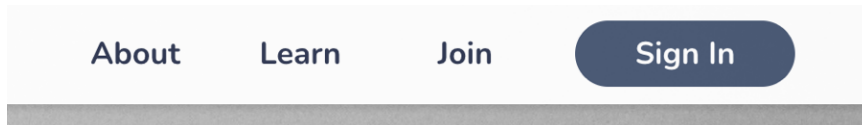
Lesson 1: TunePad

We will start with the online environment TunePad.

- 1) Open your web browser and go to: <https://tunepad.com/>
- 2) Scroll down and look at the video (see the image):



Before we really start with TunePad you need to login in order to save your tunes. You can first register or use your gmail account.



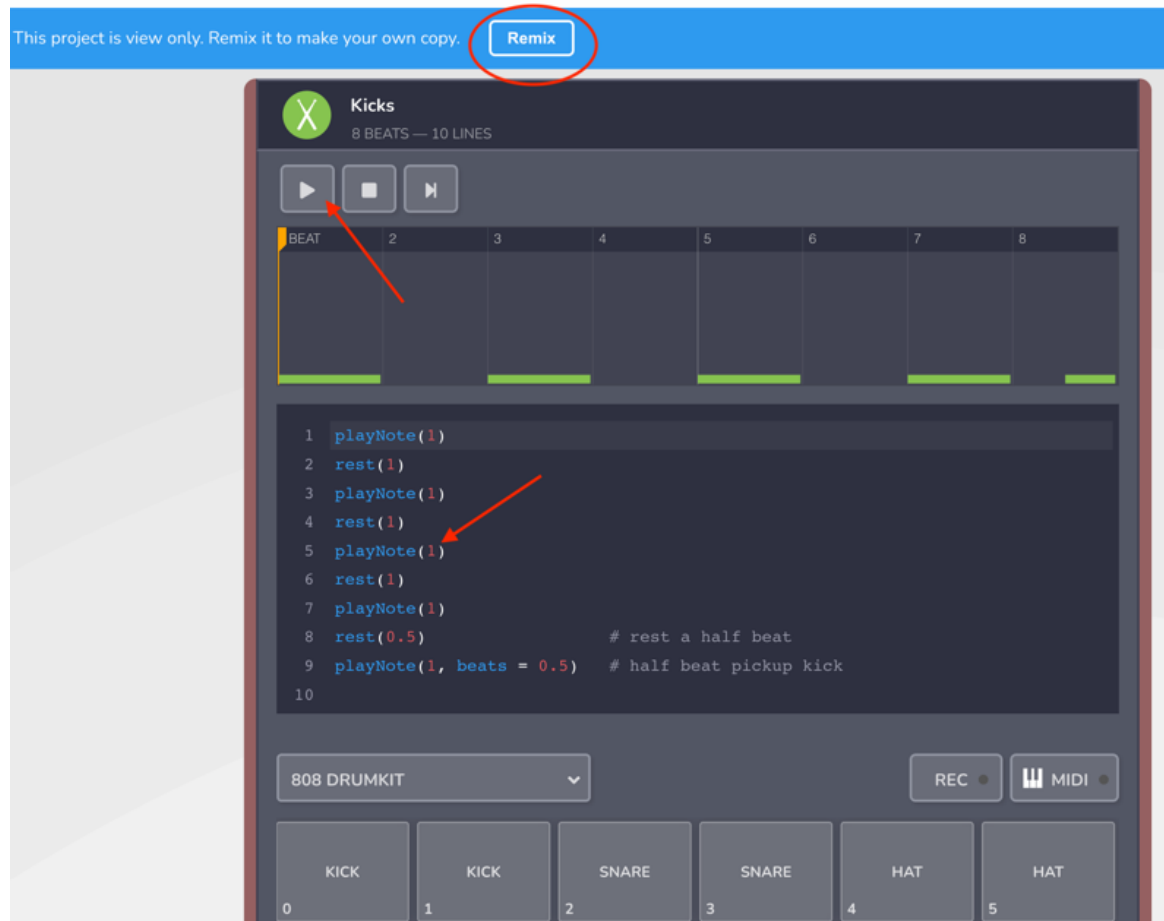
Now you are ready to start with TunePad. Look closely at the code underneath. This is Python. Do you have any idea what is happening?

The text after **#** are **comments**. Python does not do anything with this text. Programmers add comments to inform the human reader on what is happening.

```
1 playNote(1)
2 rest(1)
3 playNote(1)
4 rest(1)
5 playNote(1)
6 rest(1)
7 playNote(1)
8 rest(0.5)           # rest a half beat
9 playNote(1, beats = 0.5) # half beat pickup kick
10
```

playNote() and *rest()* are called **functions**. Functions are important concepts in many programming languages. More on functions later.

- 3) You can try the code through this link
<https://tunepad.com/project/34129>
- 4) Scroll down and play the code by clicking the green triangle (see image).
- 5) Change a couple of the 1's into 2's or 3's. (To do so you first have to click 'remix'):



- 6) What is the difference between functions `rest()` and `playNote()`? Hint: just play with the functions and listen to what is happening.
- 7) What does the number between brackets do? (e.g. `rest(1)` en `playNote(1)`)?

IMPORTANT

You hopefully noticed that the code is running in a sequential order and when coming to the end it starts over (a loop). This loop is an automatic part of TunePad. Soon you will find out how loops work in Python.

You might know that music has beats. Much of the music you hear on the radio has four (4) beats in a bar. If you count with the music, counting to 4 (and start over again) seems 'natural'. Often there is some emphasis on the first beat.

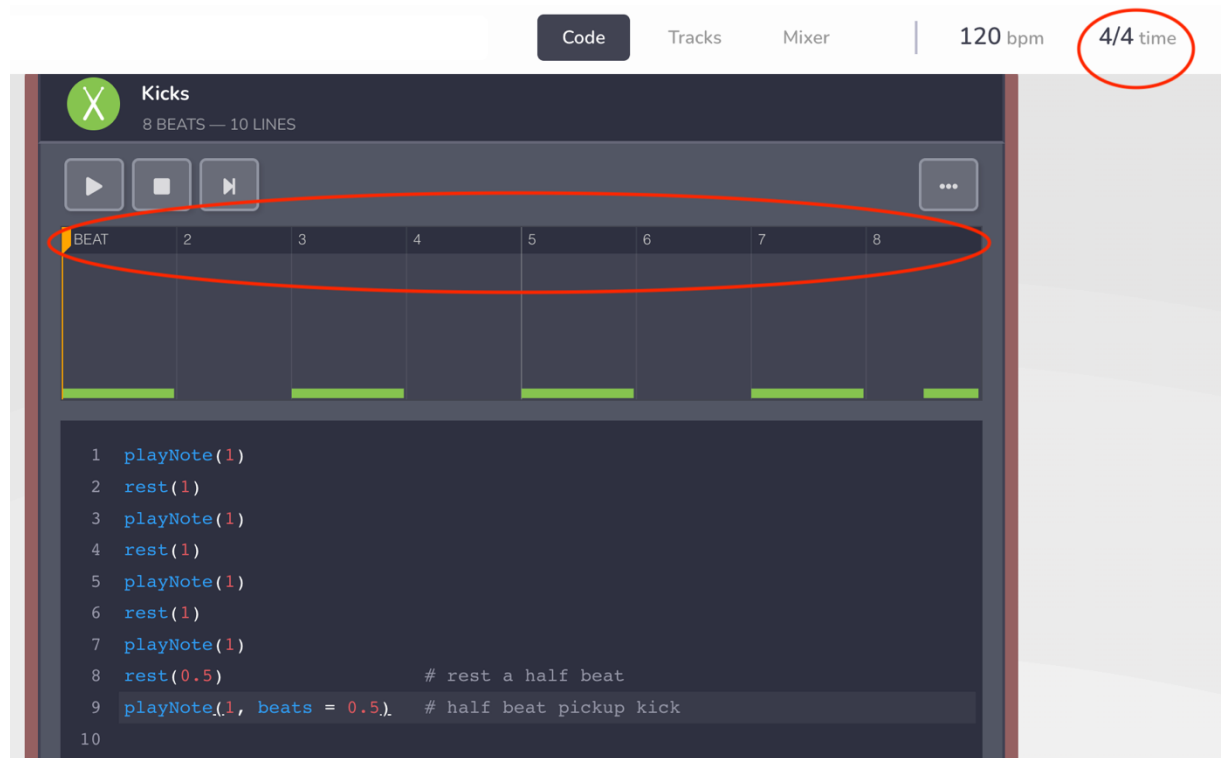
You can count to 4 with the rhythm: <https://youtu.be/lQnG9HIWok0>

Try to put a 1 on each stronger drumbeat.

However, there is also a lot of music that has three (3) beats (and there are more options such as 2 or 6)

In TunePad you need to explicitly state how many beats per bar. This is done on the right top. See the image, it says '4/4 time'.

The tune you were working with has eight beats (2 bars with 4 beats). TunePad loops after the eight beats. You can see it in the image but you can also count the numbers between brackets in the functions (1+1+1+1+1+1+0.5+0.5):

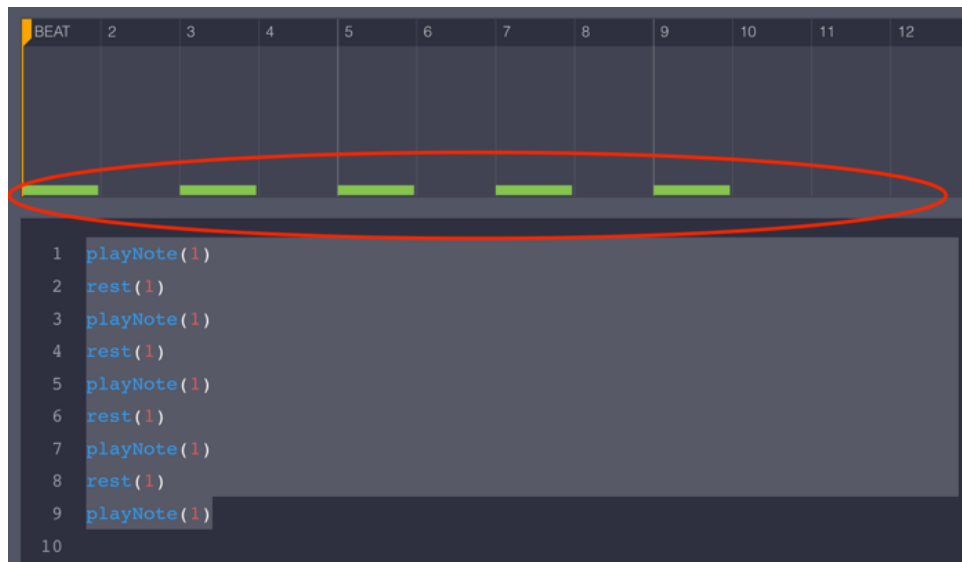


What do you think will happen if the tune would be nine beats? There is only one way to know for sure → Try out it!

```
playNote(1)
rest(1)
playNote(1)
rest(1)
playNote(1)
rest(1)
playNote(1)
rest(1)
playNote(1)
```

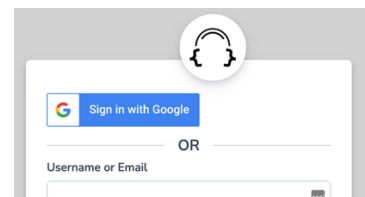
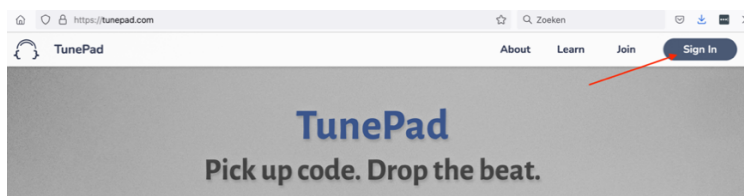
Do you understand what is happening?

TunePad has now a 4 beat to the bar rhythm. It can only loop after a complete bar (so after 4, 8, 12 etc.).



Introduction to TunePad

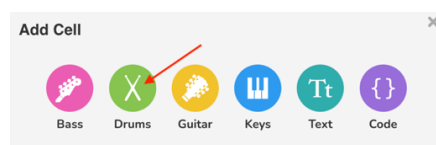
Your aim might be to learn Python, but of course you also want to create your own beats and tunes. As before, go to the website and sign in:



If you are signed in already, you can click on 'My Tunes':

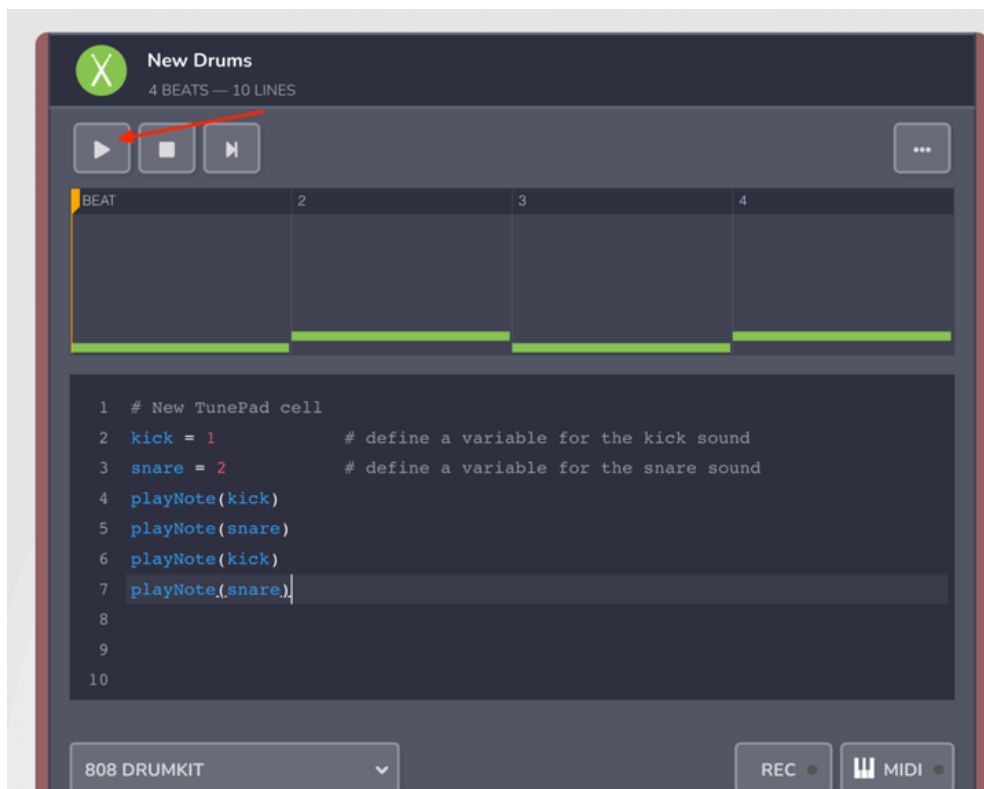


After signing in you can start a new project; click on *New Project*, click on +. You might need to add a new cell (see images):



The cell you want is *Drums*:

Add the code underneath and execute it by clicking on the triangle:



In the previous chapter the code said: *playNote(1)* instead of *playNote(kick)*. Change *kick* into *1*. Do you hear any difference? What if you change the word *kick* everywhere into *boom*? Does the code still work?

Exercise 1.1

Create your own rhythm in 2 bars (8 beats). Use at least *kick*, *snare* and *tom*. The kick sound has number 0 or 1, snare sound number 3, and tom numbers 6, 7 or 8. (see image) Also use *rest(1)* and *rest(0.5)* functions.

Python intermezzo 1

The aim of this course is learning about Python. Python was first released in 1991 by Guido van Rossum, at that time working for CWI in Amsterdam. Python has developed into one of the most used programming languages.

Guido Van Rossum



Photo: Doc Searls

Source: <https://commons.wikimedia.org/w/index.php?curid=45984776>

In this course we use the TunePad online Python editor to work with, however it is important to know there are many other **IDE**'s (Integrated Development Environments) which can be used for Python programming. Examples are IDLE and PyCharm, which need to be installed on your computer. There are also online coding environments such as Google Colab and Jupyter. In the Python Intermezzo parts of this syllabus you need to use one of such general Python editors. The examples are created with the Jupyter online notebook, but you are encouraged to experiment with other editors.

For music creation we use TunePad editor, as it has built-in **functions** for making music (such as *playNote()*). A function is a block of code that is only executed when it is called. Later in this intermezzo it is explained further.

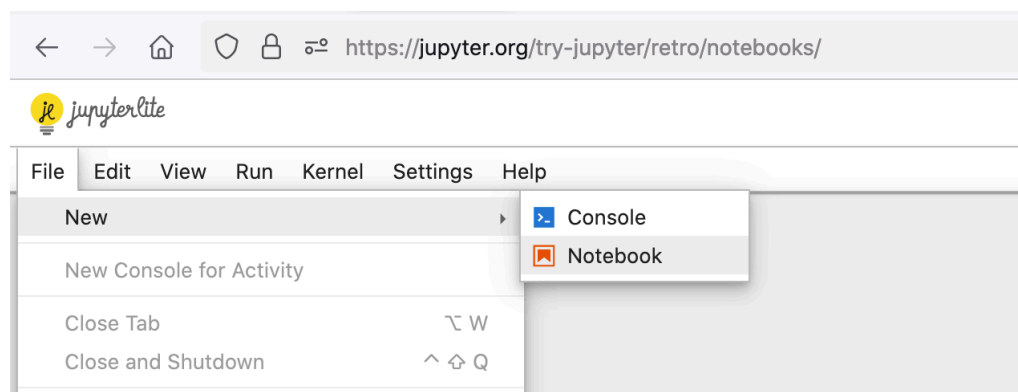
In this chapter you have already practiced with the following programming concepts:

- **Comments** → if you put # in front of a line, Python will not do anything with it. This is handy to explain in plain English what is happening in this part of the code.
- **Functions** → such as *rest()* and *playNote()*.
- **Variables** → you used variable *boom* in the line *boom = 1*. This line means that the number (integer) 1 is assigned to variable *boom*.
- **Sequence** → you might have noticed that Python seems to do things in sequence; starting with line 1 and then moving from line to line until the end of the code.

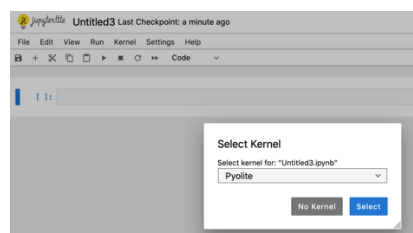
In this intermezzo you practice with these concepts outside the music context.

Step by step:

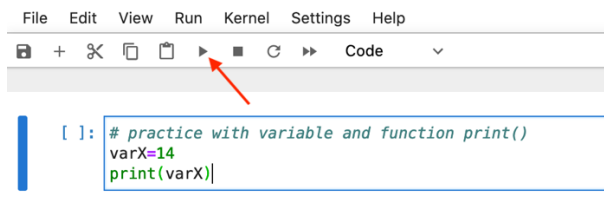
- 1) Open your general Python editor. If you use Jupyter go to <https://jupyter.org/try-jupyter/retro/notebooks/>



- 2) In case you get a pop-up about the kernel, you select *Pyolite*.

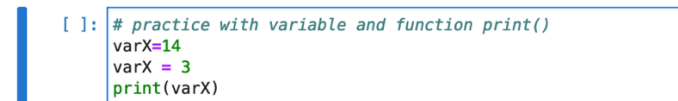


3) Try out the code underneath. What is the result when you click the triangle?



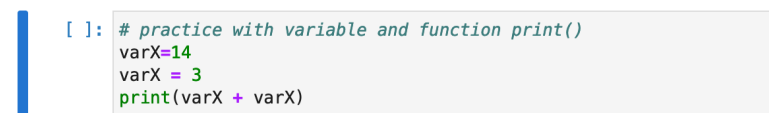
```
[ ]: # practice with variable and function print()
varX=14
print(varX)
```

4) And what is the result of this code?



```
[ ]: # practice with variable and function print()
varX=14
varX = 3
print(varX)
```

5) And this code?



```
[ ]: # practice with variable and function print()
varX=14
varX = 3
print(varX + varX)
```

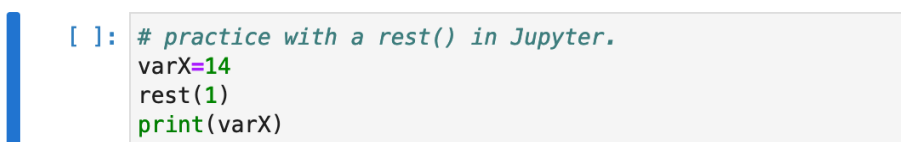
Do you understand better variables (such as varX) and how a value is assigned to a variable?

Functions

You have now used functions such as `print()`, `playNote()` and `rest()`. A function is a block of code that can perform a specific task. Examples are:

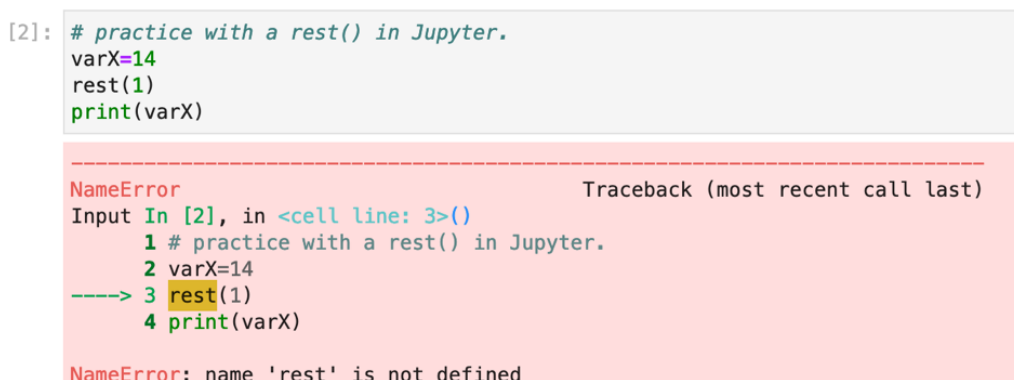
- printing something on the screen → `print()`,
- creating a sound → `playNote()`, or
- wait for a certain period → `rest()`.

Not all functions are available in all Python editors. You can try this:



```
[ ]: # practice with a rest() in Jupyter.
varX=14
rest(1)
print(varX)
```

You probably get an error message as the function `rest()` is not known in Jupyter (it is a specific NotePad function):



```
[2]: # practice with a rest() in Jupyter.
varX=14
rest(1)
print(varX)

-----
NameError                                Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 # practice with a rest() in Jupyter.
      2 varX=14
----> 3 rest(1)
      4 print(varX)

NameError: name 'rest' is not defined
```

- 6) You need to know that you can create your own functions. In this example a function is created with the name *one_print*. After that the function is carried out thrice:

```
[3]: # create function one_print()
def one_print():
    print(1)

one_print()
one_print()
one_print()

1
1
1
```

- 7) A function can also be called with a parameter between the brackets. The parameter can subsequently be used in the function, Here an example with parameter *varNum*:

```
[6]: # create function number_print()
def number_print(varNum):
    print(varNum)

number_print(3)
number_print(5)
number_print(7)

3
5
7
```

Do you understand what happens? First *number_print(3)* is called. In the function the number 3 is assigned to the variable called *varNum*. Later *varNum* is printed.

Functions are not an easy subject. In many tutorials it is only addressed after a couple of chapters. However, as functions are used right from the start in TunePad it is necessary to introduce them already in lesson 1.

Important to know:

Python is very strict with **indents**. You see that a function is constructed with the keyword *def*.

After the name and the parameter there is a **:**

All lines that are part of the function need to be indented in the same manner. The code underneath is not going to work:

```
[ ]: # create function number_print()
def number_print(varNum):
    print(varNum)

number_print(3)
```

What kind of error message do you get after executing this code? Do you understand the error message? Can you fix it? (hint: have a look at the code above).

In this intermezzo you practiced with comments, variable assignments and functions. You learned how to create functions and to interpret some error messages. Also you now know about indentation in Python. That's a lot.

Exercise 1.2

Create a Python function with the name *double_number*. This function takes a number and doubles it. The result is printed on screen.

Book reference

- Read chapter 1: Why music and coding.
- Carry out the steps in Interlude 1: Basic pop beat

Assignment 1

All assignments need to be handed in in the DLO (<https://dlo.mijnhva.nl>)

You can use the function `playNote()` in the following manner:

```
playNote(1)
```

but also:

```
playNote(1, beats=0.5)
```

`playNote()` can take only one parameter, but more is also possible. If the *beats* parameter is not specified, the default duration of 1 is taken.

Create a function *beatRhythm()* that takes a parameter.

In the function a note is played. The duration of the note is the parameter that is received.

Call the function at least 8 times (with various durations) so that it creates a nice rhythm.

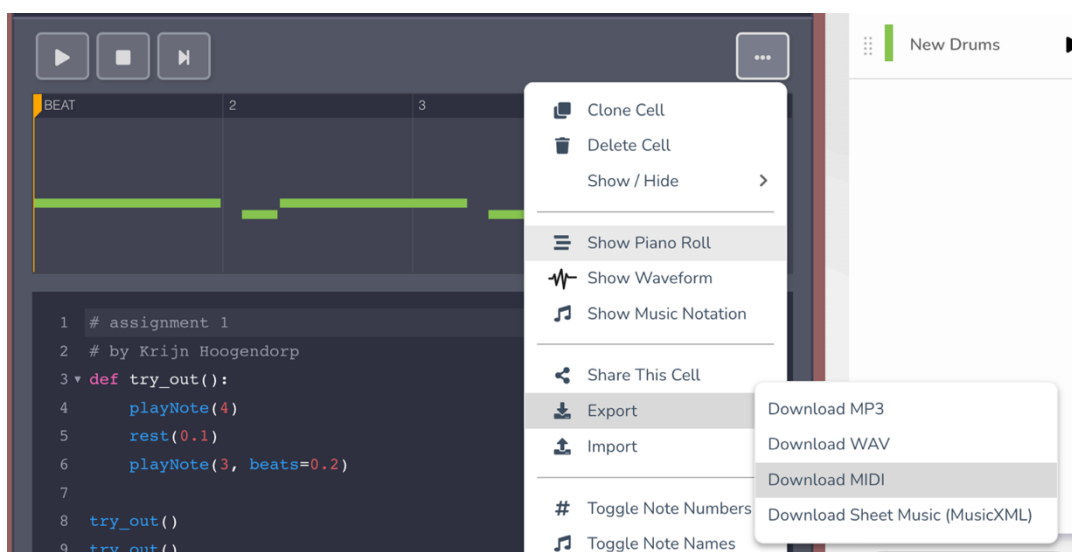
Handing in the assignment.

You have the hand in the assignment at the DLO. You probably can find the course here: <https://dlo.mijnhva.nl/d21/home/443650>

However, you can also go to: <https://dlo.mijnhva.nl/>

And look for the course Learning Python through coding music.

You hand in the MIDI file: go to 'Export' and click on 'Download MIDI'



When submitting the assignment you put the URL of the TunePad code in the comments and upload the MIDI file:

The image shows a 'Submit Assignment' form on the left and a file upload modal on the right. The form has a 'Files *' section with '(0) file(s) to submit'. Below this, it says 'After uploading, you must click Submit'. There are three buttons: 'Add a File', 'Record Audio', and 'Record Video'. A red arrow points from the 'Add a File' button to the 'Comments' section. The 'Comments' section has a text area with a red circle around the URL 'https://tunepad.com/project/37034'. The modal on the right is titled 'Add a File - Learning Python through coding music Amsterdam'. It contains a dashed box with the text 'Drop files here, or click below!' and an 'Upload' button. Below the dashed box, it says 'You can upload files up to a maximum of 3 GB.' At the bottom of the modal are three buttons: 'Add', 'Back', and 'Cancel'.

Submit Assignment

Files *

(0) file(s) to submit

After uploading, you must click Submit

Add a File Record Audio Record Video

Comments

Paragraph **B** *I* U **A**

<https://tunepad.com/project/37034>

Add a File - Learning Python through coding music Amsterdam

Drop files here, or click below!

Upload

You can upload files up to a maximum of 3 GB.

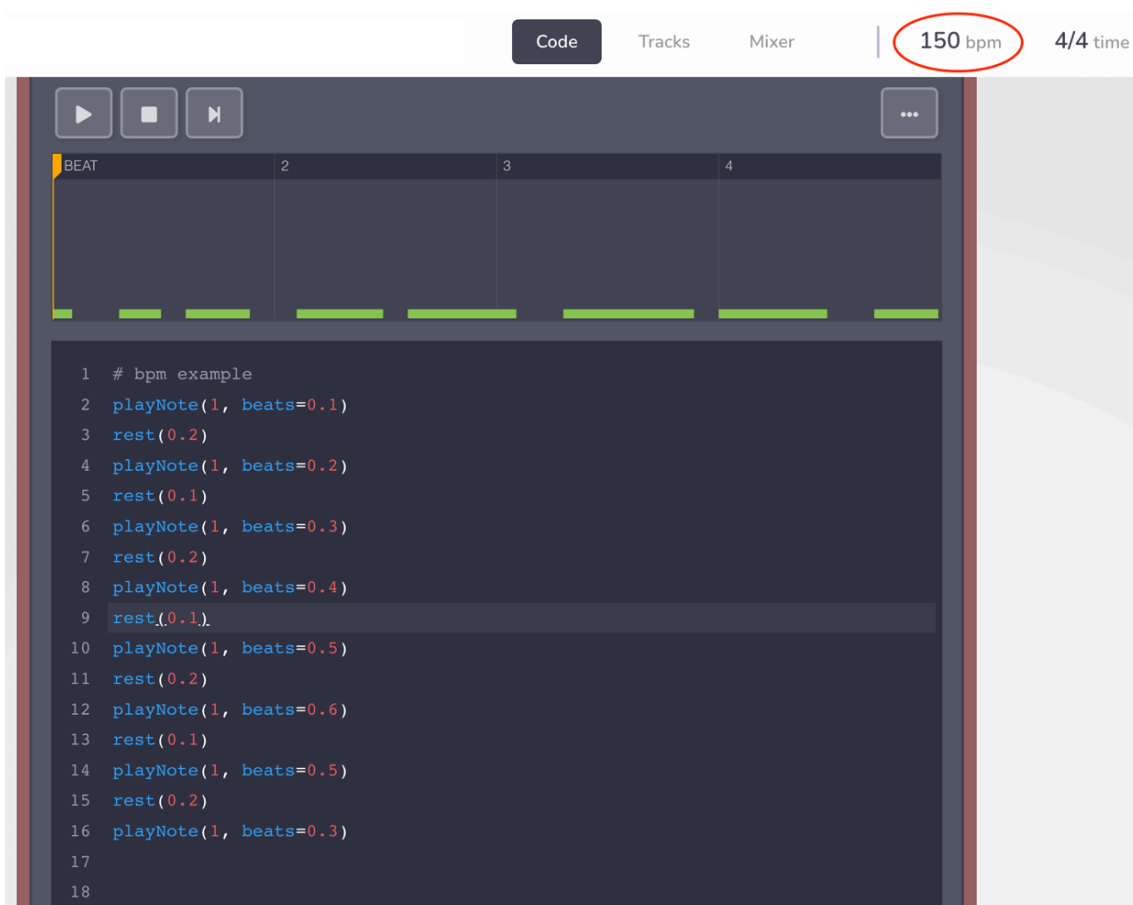
Add Back Cancel

Lesson 2: Tempo and rhythm

If you didn't know before, lesson 1 has taught you that music is, among other things, defined by beats. You have read (and maybe experienced) that a 4 beats per bar is quite common.

You probably know already that another factor is the speed in which the rhythm is played. This speed is usually indicated by *bpm* (beats per minute). In general, rock music is around 100 to 140 bpm, while R&B is a bit slower with 60 to 80 bpm. Techno on the other hand, is more up tempo with 140 bpm and over.

You can change the bpm of you TunePad creation on the top of the cell.



Exercise 2.1

Take the TunePad code of the image above and change the bpm into 70 and then into 100. Experience the difference.

Beats per bar

In TunePad, on the right of bpm, you see the beats per bar. In the image it says 4/4. The top number tells you how many beats there are in a single bar.

Understanding the bottom number is a bit more difficult for people with limited musical background.

If the bottom number is a 4, it means the beats are quarter notes (four quarter notes in a bar).

If the bottom number is 2, it means the note value is half notes (half notes per bar).

Do not worry so much about understanding the bottom number.

4/4 time means 4 beats to the bar.



Exercise 2.2

Take your tune of the previous exercise and play with the time (e.g. change it into 3 /4 or 6/8). What do you notice?

Loops

A very nice and flattering image of programmers is related to their laziness which drives them to create efficient code (meaning 'less' code). Take for example the two blocks of code underneath:

A screenshot of the TunePad code editor. The code is as follows:

```
1 # no loop
2 playNote(1, beats=0.5)
3 rest(0.5)
4 playNote(2, beats=0.5)
5 rest(0.1)
6 playNote(1, beats=0.5)
7 rest(0.5)
8 playNote(2, beats=0.5)
9 rest(0.1)
10 playNote(1, beats=0.5)
11 rest(0.5)
12 playNote(2, beats=0.5)
13 rest(0.1)
14 playNote(1, beats=0.5)
15 rest(0.5)
16 playNote(2, beats=0.5)
17 rest(0.1)
18 playNote(3)
```

The interface shows a piano roll at the top with green bars representing notes and rests. At the bottom, there are controls for '808 DRUMKIT', 'REC', and 'MIDI'.A screenshot of the TunePad code editor. The code is as follows:

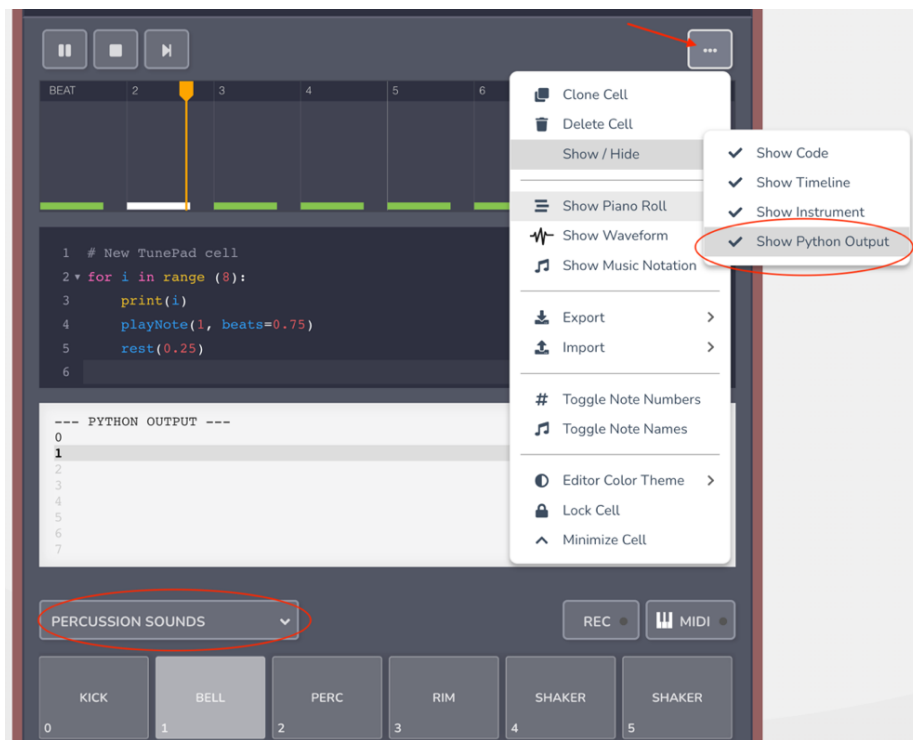
```
1 # with a for loop
2 for i in range(4):
3     playNote(1, beats=0.5)
4     rest(0.5)
5     playNote(2, beats=0.5)
6     rest(0.1)
7     playNote(3)
8
9
10
11
12
13
14
15
16
17
18
```

The interface shows a piano roll at the top with green bars representing notes and rests. At the bottom, there are controls for '808 DRUMKIT', 'REC', and 'MIDI'.

If you try out both blocks of code, you notice that the result is exactly the same. Even people illiterate in coding would probably prefer the shorter version with the for-loop.

A *for-loop* carries out a specific block of code a number of times. You can try out the code underneath. You need to make some changes to the TunePad configuration:

- 1) This time we want to hear the 'cowbell' which is part of the percussion sounds.
- 2) There is a `print()` statement in the code. We want to see the result. Click on the ... and 'Show/Hide' and enable 'Show Python Output':



Explanation of code:

- function *range(8)* gives you list of 8 numbers starting with 0: [0,1,2,3,4,5,6,7]
- *for i in range(8)* means that the program will step through the list (first 0, then 1 etcetera).
- after every step the indented lines are carried out.

Try out the following code:

```
for i in [0,1,2,3,4,5,6,7]:
    print(i)
    playNote(1, beats=0.75)
    rest(0.25)
```

Is there any difference with the previous code?

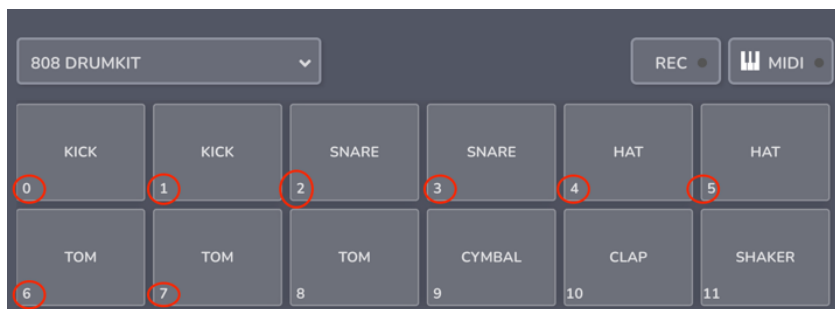
Exercise 2.3

Create a program that continuously plays:

- a kick followed by the other kick sound, then
- a snare followed by the other snare, then
- a hat followed by the other hat, and
- finally a tom followed by the other tom sound.

You cannot use more than 4 lines of code.

Hint: use a loop and have a look at the image underneath:



Python intermezzo 2

In chapter 1 you have already practiced with functions and variables. You also have seen how comments work and practiced with error messages. In lesson 2 lists and for-loops were added to your knowledge.

In this intermezzo a little more is explained on lists and loops.

Lists and other datatypes

You have seen that the function `range(8)` can have a somehow similar use as *list* `[0,1,2,3,4,5,6,7,8]`

A list is usually put in between square brackets: `[` and `]`. A list is actually a datatype. Within programming all data has a specific type.

For example in the line `varA = 3`. The value of variable `varA` is an integer (3), thus the datatype of `varA` is an integer.

In case of `varB = [0,1,3]`. The value of `varB` is `[0,1,2,3]` and the datatype is list.

You can check the datatype of a variable with the function `type()`. Try the following in your favorite Python editor:

```
varA = 3
print(type(varA))
```

```
varB = [0,1,2]
print(type(varB))
```

You will see that the datatype of `varA` is an *integer* (`int`) and the datatype of `varB` is a *list*.

But what happens if you try the following code?

```
varX = "hallo"
print(type(varX))
```

As you can see, the datatype of `varX` is `'str'` this is short for *string*. A string can contain 0 or more characters.

Exercise 2.3

Try out the code underneath and explain the two results of the print-statements:

```
varInt = 3
varStr = "4"
print(varInt + varInt)
print(varStr + varStr)
```

And what about executing the code underneath? Do you understand what is happening?

```
varInt = 3
varStr = "4"
print(varInt + varStr)
```

Book reference

- Read chapter 2: Rhythm and temp.
- Carry out the steps in Interlude 2: Custom trap beat.

Assignment 2

Have a look at (part of) the following music video of the song Lowrider by War:

<https://youtu.be/BsrqKE1iqqo>

A key percussion sound in the song is the cowbell. Other famous ‘cowbell’ songs are:

- Don’t fear the reaper – Blue Oyster Cult (<https://youtu.be/Dy4HA3vUv2c>) which inspired this sketch (<https://youtu.be/R8fpVNhiqKQ>)
- Honky Tonk Women – Rolling Stones (https://youtu.be/hqqkGxZ1_8I)

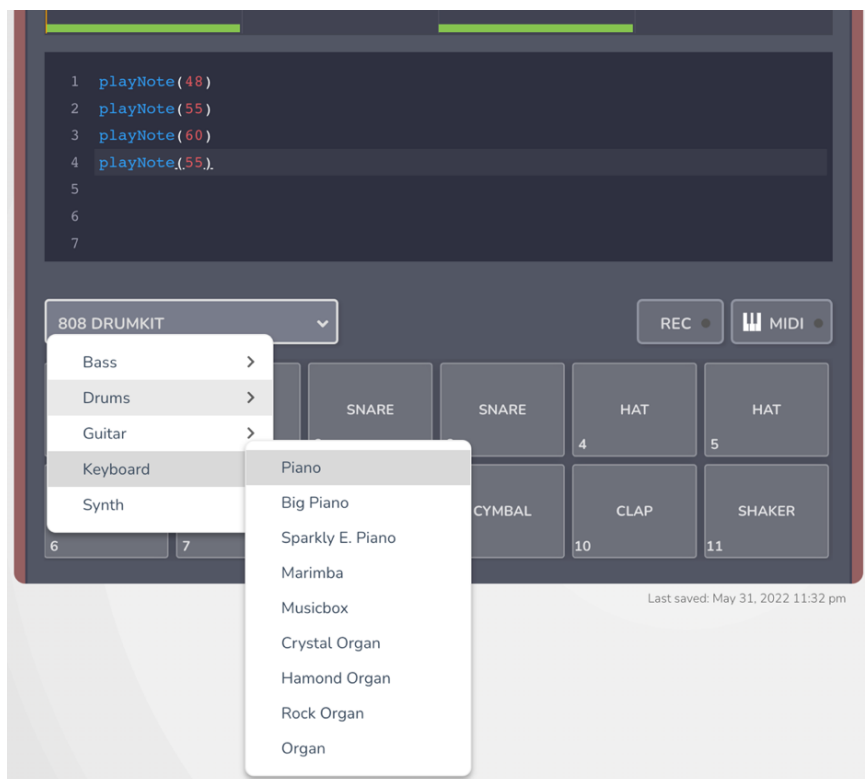
In this assignment you need to use TunePad to program the cowbell rhythm in Lowrider. It is possible to use the following video as help: <https://youtu.be/ShsSpqt2IVE>

Try out the different percussion sounds.

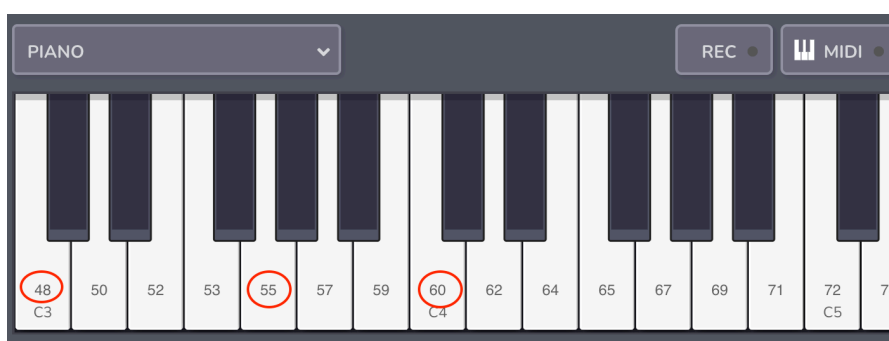
Go to the DLO and hand in the MIDI file and the URL

Lesson 3: TunePad melody

Of course, music is more than just rhythm, it also consists of melody. TunePad holds different musical instruments. Try out the piano with the code underneath:



You can try this melody also out on an actual instrument (if you have one available): `playNote(48)` gives a C note, `playNote(55)` a G note, and `playNote(60)` also a C note (but an octave higher).



The numbers used are so-called **MIDI** note numbers. MIDI is a protocol that can be used to exchange musical information between digital systems.

You can change the pitch by changing the MIDI-numbers. Try the same rhythm with numbers: 55, 62 and 67.

As with drums, it is possible to add effects to the piano. Try the code underneath. What do *beats* do?

```

1 playNote(36, beats = 0.5)
2 playNote(36, beats = 0.5)
3 playNote(43, beats = 0.5)
4 playNote(43, beats = 0.5)
5 playNote(48, beats = 0.5)
6 playNote(48, beats = 0.5)
7 playNote(43, beats = 0.5)
8 playNote(43, beats = 0.5)
9

```

As drums, also piano sounds can have effects. Try the code underneath. You know what beat does, but what does *velocity* do?

```

for i in [36, 43, 48, 43]:
    playNote(i, beats=0.5, velocity=25)
    rest(0.1)
    playNote(i, beats=0.5, velocity=50)
    rest(0.3)

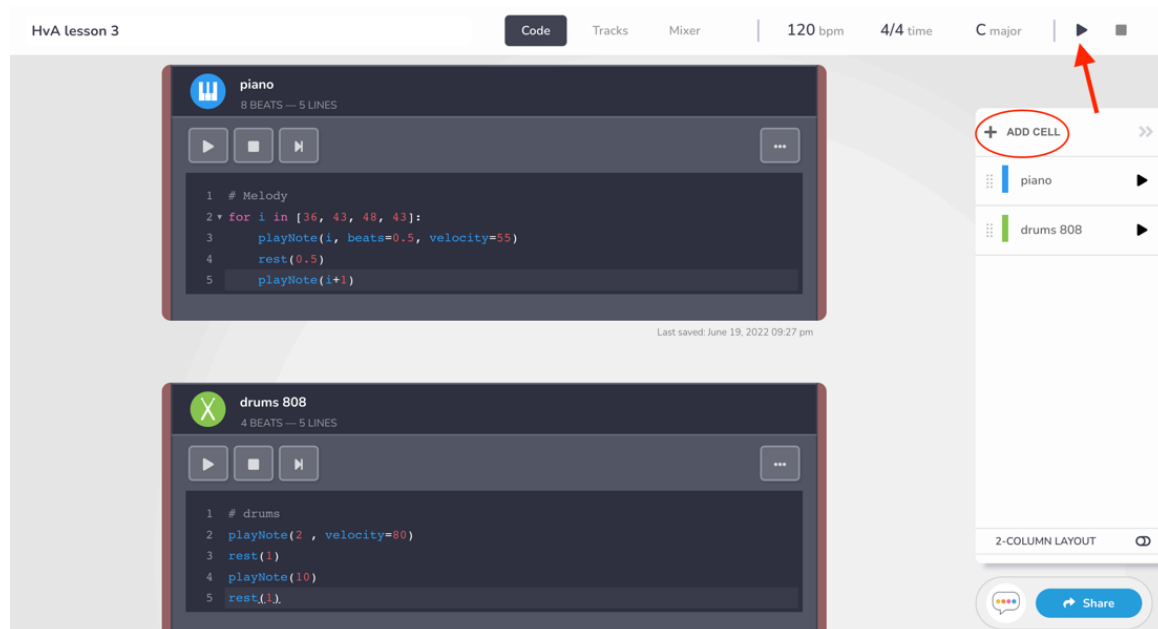
```

Explain how the *for* construction works in the code above.

Music with different layers

‘Real’ music compositions usually have a variety of sounds. Traditional rock is often a combination of guitar, bass and drums. With TunePad it is possible to join different sounds.

To do this you have to add different cells to a project and with the triangle right on top you can start these different cells at the same time.



Exercise 3.1

Create your own song containing piano and percussion.

Python intermezzo 3

In Python intermezzo 2 you have read that each variable has a certain datatype. Within Python it is not necessary to explicitly state in advance which datatype a variable has. In some other computer languages this might be different. For example, in the programming language Java, a variable (`varX`) can be declared as such:

```
int varX = 10;
```

`varX` has thus datatype integer (`int`).

In Python you can write:

```
varX = 10
```

As 10 is a whole number (not a fractional number) Python assumes the datatype will be an integer. You can try out the following code in your favorite Python editor:

```
varX = 10
print(type(varX))

<class 'int'>
```

Frequently used simple datatypes in Python are: integer, float, strings and booleans:

- **integer** – whole numbers (not fractionals) such as 10, 2015, and -13
- **float** – decimal numbers, such as 0.5, 14.234, and – 12.123
- **string** – a set of characters, such as: “b”, “2@#4x”, or “__appelx 3s”
- **boolean** – a type that can only have the values *True* or *False*.

Exercise 3.2

Use the following code to find the datatypes of the variables.

```
varA = 10
varB = "10"
varC = "True"
varD = True
varE = 10.0
varF = "10.0"
print(type(varA))
print(type(varB))
print(type(varC))
print(type(varD))
print(type(varE))
print(type(varF))
```

Lists

A list (such as `[1, 14, -1]`) is one of the more complex datatypes. It is complex as the list contains elements that also have their own specific datatype. The list `[1, 14, -1]` contains elements with datatype integer.

Each item in a list corresponds to an index number, which is an integer value, starting with the index number 0.

In the code underneath, the first element (1) has index number 0.

Consequently the fourth element (4) has index number 3.

You can print out a single element by using the index number:

```
listA = [1,2,3,4]
print(listA[0])
```

1

A list can have elements of different types. Take this list for example:

```
varListA = [1, "hallo", True]
```

The datatype of varListA is list (you can try *print(type(varListA))*).

However, the first element (which has index 0) is of datatype *integer*.

The following element, *"hallo"*, is a string

And the last element, *True*, is a boolean.

Do not take our word for it. Check it yourself:

```
varListA = [1, 'hallo', True]
print(type(varListA))
print(type(varListA[0]))
print(type(varListA[1]))
print(type(varListA[2]))
```

List manipulation

In the previous intermezzo you have seen what happens if you 'add' strings in comparison with adding integers.

```
varInt = 3
print(varInt+varInt)
varStr = "hallo"
print(varStr + varStr)
```

6

hallohallo

Adding lists is similar to adding strings; they are linked together (concatenated).

```
listA = [1,2,3]
listB = [2,1,4]
listC = listA+listB
print(listC)
```

[1, 2, 3, 2, 1, 4]

Another fine trait of lists is that they can be changed after creation (they are mutable). E.g. with the function `append()` an element can be added.

```
listA = [1,2,3,4]
listA.append(5)
print(listA)
```

```
[1, 2, 3, 4, 5]
```

Exercise 3.3

If you have `listX = [1,2]` and `listY = [2,3]`

What is the difference between `listX+listY` and `listX.append(listY)`?

As you might have gathered from the previous question, it is possible to have a list in a list. Another reason to consider lists as a *complex* datatype:

```
listX = [1,2,[3,4]]
```

In the above example, the element with index 2 is a list ([3,4]):

```
listX = [1,2,[3,4]]
print(listX[2])
```

```
[3, 4]
```

So how, to find a single element in such a list in a list? You can use the following syntax:

```
listX = [1, 2, [3, 4]]
print(listX[2][1])
```

```
4
```

Exercise 3.4

So how to get element “red” from `varList` underneath:

```
varList = ["A", "B", [1, 2, ["red", "blue"]]]
```

A list is just one of the complex datatypes of Python. Some others are:

- **Tuples**, which resemble lists, but after creation they cannot be adapted (they are immutable).
- **Dictionaries** which can hold key:value combinations. To find the value, you can reference the key.

Here examples of the use of tuples and dictionaries. Do you understand what is happening?

```
#This is an example of a tuple
varTupleA = ("bear", "fox")
print(type(varTupleA))
print(varTupleA[0])

#This is an example of a dictionary
varDictA = {"NL":"Amsterdam", "FR":"Paris"}
print(type(varDictA))
print(varDictA["FR"])

<class 'tuple'>
bear
<class 'dict'>
Paris
```

Book reference

- Read chapter 3: Pitch, harmony and dissonance
- Carry out the steps in Interlude 3: melodies and lists.

Assignment 3

In this assignment you are going to add to the work that has been done in Interlude 3 of the book (Melodies and lists on page 68). The subject of the interlude is Beethoven's composition *Für Elise*.

https://youtu.be/s71I_EWJk7I

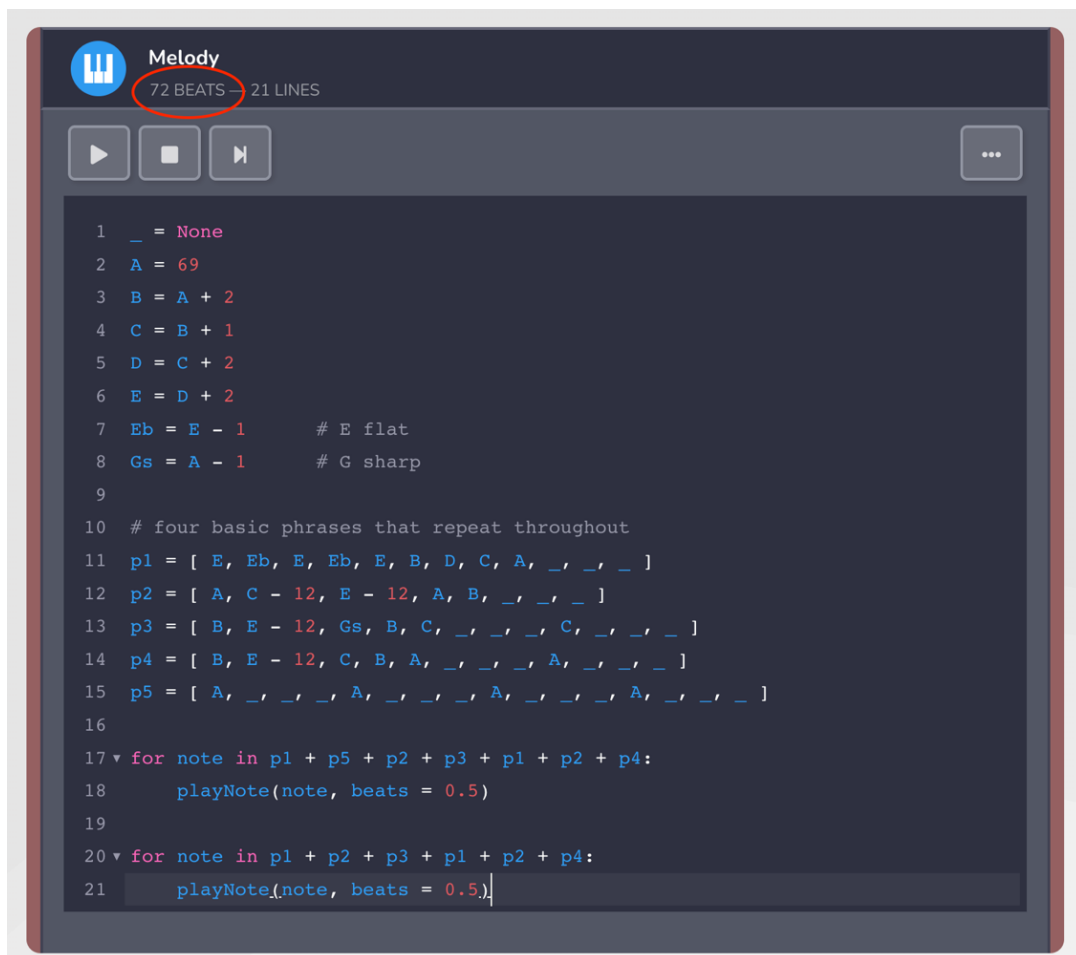
In the interlude a remix of *Für Elise* by the YouTuber Kyle Exum is presented.

<https://youtu.be/EWYAQHMIa4Y>

In the remix the traditional piano melody of *Für Elise* is combined with a modern bass and drums.

An interesting aspect of the remix is that the complete piano piece is 72 beats before it loops back to the start. The drums and bass are 76 beats. This means that they are not synchronous. Each loop the combination creates a slightly different effect.

Please have look at the code. Underneath it will be explained:



In lines 1 to 8 variables are defined. Actually each variable name is a note. The variable `_` (on line 1) means that there will be no sound (None).

In lines 11 to 16 five lists are defined. Each lists is a sequence of notes:

- P1 has 12 notes
- P2 has 8 notes
- P3 has 12 notes
- P4 has 12 notes
- P5 has 16 notes

From line 17 to line 21 there are 2 for-loops.

The first for loop combines 7 of the lists. In total there are 80 half notes (each with beat=0,5)

The second loop combines 6 lists with a total of 64 half notes

The combination of 80 half notes and 74 half notes gives the total of the previous mentioned 64 beats.

The drum code is much simpler than that. It is a rest of 12 beats followed by 16 times a sequence of 5 sounds:

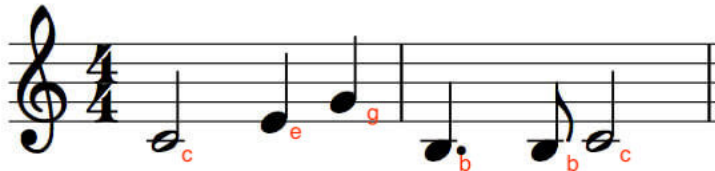
```
rest(12)
for i in range(16):
    playNote(kick)
    playNote(snare, beats = 0.5)
    playNote(snare, beats = 0.5)
    playNote(clap)
    playNote(kick)
```

In this assignment you need to make your own drum sequence. The code must contain at least three lists that can be combined in different ways.

Do not forget to hand in the assignment (as a link to TunePad and as a MIDI file).

Lesson 4: Chords

In lesson 3 you have learned about melody and how lists can assist in playing a melody. This is a simple melody



In MIDI format:

c = 60
e = 64
g = 67
b = 59
b = 59
c = 60

```
playNote(60, beats = 2)
playNote(64, beats=1)
playNote(67, beats=1)
playNote(59, beats=1.5)
playNote(59, beats = 0.5)
playNote(60, beats=2)
```

You might want to code this with a for-loop that is iterating through a list. However, not all notes have the same length. How to deal with such a problem?

For this you need to develop real coding skills. There are probably a variety of solutions, but this is one of them:

You noticed already that each note has a pitch (such as 48) and a length (e.g. 2). These attributes can be combined into a list of two elements: [48, 2]. With this simple melody you get 6 small lists: [60,2], [64,1], [67,1], [59,1.5], [59, 0.5], and [60, 2].

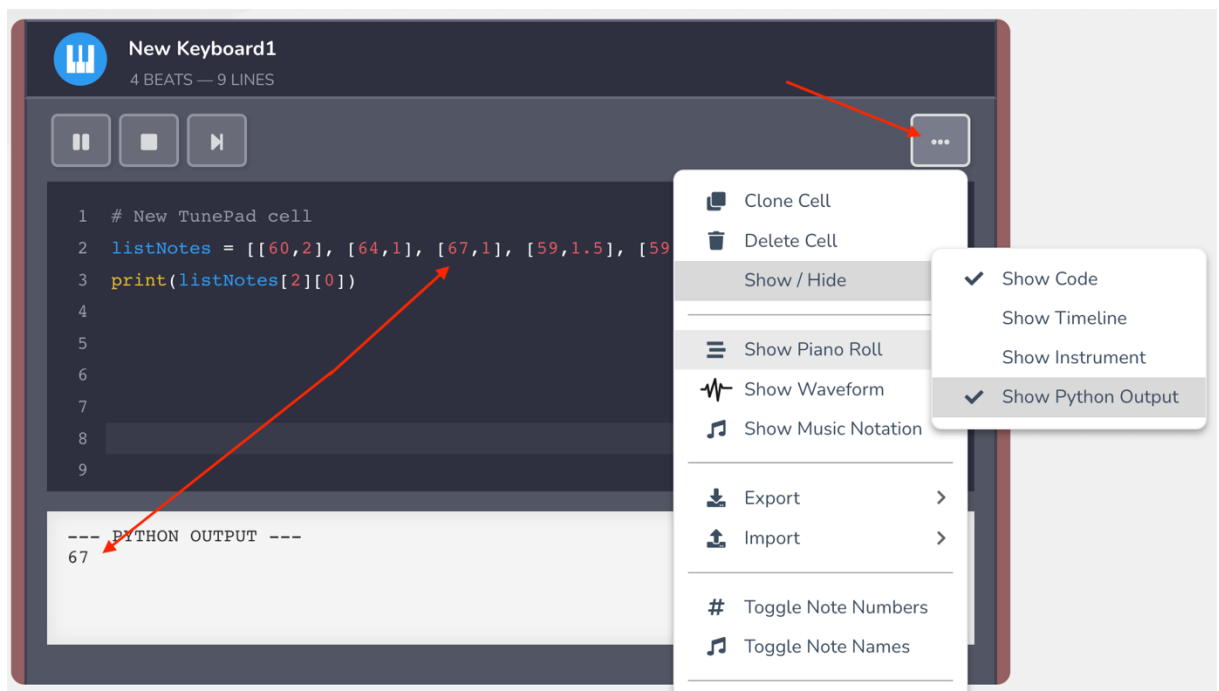
These smaller lists can be combined in one bigger list

```
listNotes = [[60,2], [64,1], [67,1], [59,1.5], [59, 0.5], [60, 2]]
```

Now we are getting somewhere.

As you learned in the Python intermezzo of lesson 3, you can identify each element with Python.

Try out the following code:



And if you want, you can play a note like this:

```
playNote(listNotes[3][0], listNotes[3][1])
```

Exercise 3.5

Put all notes in a list (as above) and use the info in this paragraph to code the simple melody with a for-loop.

Exercise 3.6

There are other ways to code the same simple melody, but now you put the pitch in one list and the duration in the other:

while using the list. Underneath you find a framework for such code. The spots with **** need to be changed to make it work:

```
listPitch = [60, 64, 67, 59, 59, 60]
listDuration = [2, 1, 1, 1.5, 0.5, 2]
```

```

listPitch = [60, 64, 67, 59, 59, 60]
listDuration = [2, 1, 1, 1.5, 0.5, 2]

for i in range(0, len(listPitch)):
    playNote(listPitch[**], ****[**])

```

So, now you have seen that there are often various solutions possible for a problem. This understanding is important for anyone who codes. Moreover you have practiced with using the representation of pitch and duration into lists, and then using the lists to recreate a melody.

These are big steps.

Chords

You now have learned about rhythm and melody. Both important aspects of music. Some melodies seem to be more pleasant to the ear than others, but do not be rigid about what melodies consist of ‘good music’ and what melodies are ‘wrong’.

Bassist and educator Victor Wooten advocates taking ‘right’ and ‘wrong’ out of the picture in creating music:

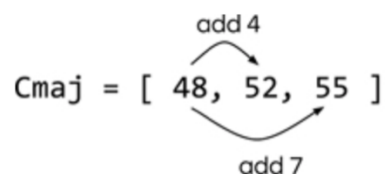
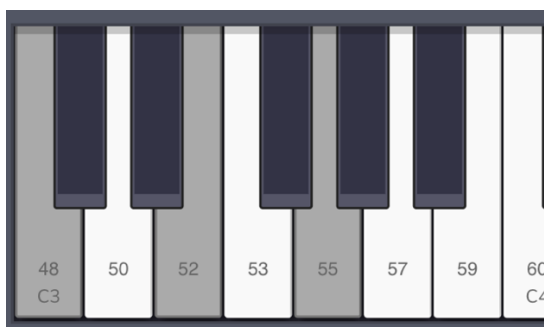
<https://youtu.be/UBNT5K2sEFQ>

So far the melody has been a succession of single notes. However, it is also possible to have notes sounding simultaneously. These are called chords.

Chords are possible in TunePad with the use of lists. In the code underneath you hear the sound of the C major chord:

```
2 Cmajor = [ 48, 52, 55 ]
3 playNote(Cmajor)
```

A C major chord, has the C note as a root note combined with an E and a G.



How will you make an F major? Now you will take 53 as a root note followed by 57 (53+4) and 60 (53+7)

```

Cmajor = [48, 52, 55]
Fmajor = [53, 57, 60]

playNote(Cmajor, beats = 0.5)
rest(0.5)
playNote(Cmajor, beats = 0.5)
rest(0.5)
playNote(Fmajor, beats = 0.5)
rest(0.5)
playNote(Fmajor, beats = 0.5)
rest(0.5)

```

Python intermezzo 4

In this intermezzo we continue the practice of functions (as introduced in intermezzo 1). A function is block of code that is executed when it is called. It can take parameters as input and can return data as a result. Underneath the example of a function named *twoTimes()* that takes a word (a string) and returns it twice.

```

def twoTimes(word):
    twice = word+word
    return twice

info = twoTimes("Amsterdam")
print(info)

```

AmsterdamAmsterdam

The nice thing about functions is that you can write code once and use it multiple times (so less work). Even better, there are a lot of functions available that have been written by others (so no work). An example are functions within the random-module.

To use the *random* module you have to import it first. Underneath you do the import and then use the function *uniform()*

```

import random
randomInteger = random.randint(0,10)
print(randomInteger)

```

7

The function *randint()* takes two integers parameters, and returns an integer that is between these two numbers.

Exercise 3.6

you might need a random decimal number (a float). For this you can use the function `uniform()`. Create code in which `uniform()` is used.

For information on the random functions you can have a look at:

<https://docs.python.org/3/library/random.html>

If you do an `import random`, and use the `uniform()` function multiple times, you need to write out `random.uniform()` each time.

To shorten it, you might want to use the import line:

`from random import uniform`

If you do that you can directly use `uniform()`

Book reference

- Read chapter 4: Chords

Carry out the steps in Interlude 4: Playing chords.

Assignment 4

In real life the rhythm of the music has some ‘randomization’. For inexperienced musicians it can be difficult to play ‘exactly on the beat’. The note might be slightly earlier or later.

The experienced musician might use timing in a deliberate way, not playing on the beat can make the piece more interesting and create a surprising effect for the listener.

In this exercise the chord F-major is not played exactly on the beat. Have a look at the code underneath and change `***` in such a way that there is a random rest between C-major and Fmajor:

```
from random import uniform

Cmajor = [48, 52, 55]
Fmajor = [53, 57, 60]
for i in range(16):
    playNote(Cmajor, beats = 0.5)
    rest(0.5)
    playNote(Cmajor, beats = 0.5)
    randomRest = uniform(***, ***)
    rest(***)
    playNote(Fmajor, beats = ***)
    rest(0.5)
    playNote(Fmajor, beats = 0.5)
    rest(0.5)
```

Lesson 5: Scales, keys and melody

In the previous lesson you have learned about chords. It was mentioned that chords have a root note. The root note of the C major chord is (logically) a C.

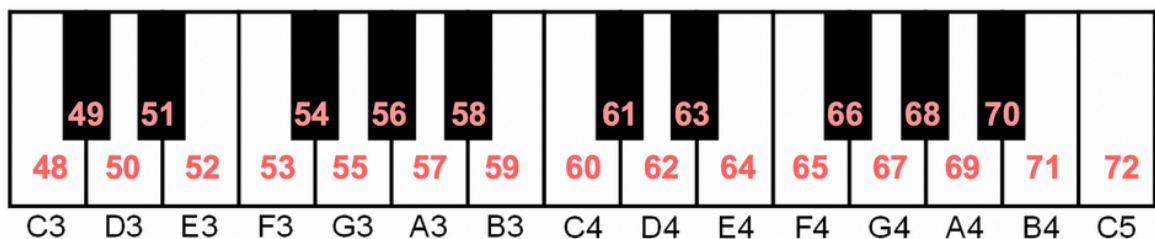
In Western music compositions usually also have a root note. This is called the **key**. Most chords and notes will then be part of the **scale** of the key.

For example, the song Hallelujah by Leonard Cohen has the key C major. All notes in the song are part of the C Major scale (C, D, E, F, G, A, B, C).

Hallelujah by Leonard Cohen:

<https://youtu.be/M6mU0wMEmbQ>

The C major scale is interesting as only the whole notes are used. Whole notes are the white keys on the piano. The black keys are half notes.



This means the C major scale is 48, 50, 52, 53, 55, 57, 59, 60

Exercise 5.1

Create a NotePad python program that plays the C major scale

Extra: you can reverse the list `CmajorScale` with `CmajorScale.reverse()` . Play with that as well.

There is another way you can define the C major scale in a list

rootNote = 48

```
majorScale = [rootNote, rootNote+2, rootNote+4, rootNote+5, rootNote+7, rootNote+9,
rootNote+11, rootNote+12]
```

With this knowledge you can easily change the root note and still have a valid major scale.

```
rootNote = 48
majorScale = [rootNote, rootNote+2, rootNote+4, rootNote+5, rootNote+7,
              rootNote+9, rootNote+11, rootNote+12]

for note in majorScale:
    playNote(note)
```

Exercise 5.2

Create a function which takes as a parameter the root node (in MIDI notation) and plays the major scale of that root node.

There are many other scales possible (besides the major scale). In Western music the minor scale is also quite common. Usually, the minor scale is considered a bit sad in comparison with the 'happy' major scale.

The C minor scale has the following notes: 48, 50, 51, 53, 55, 56, 58 and 60.
Notice that the 51, 56 and 58 are half notes.

Exercise 5.3

Create a function which takes as a parameter the root node (in MIDI notation) and plays the minor scale of that root node.

Python intermezzo 5

A very much used construct in most programming languages is the if-statement. If something is true then an action is carried out.

For example:

```
if(True):  
    print("it is true")
```

it is true

So what is between the brackets () should be true? In this case we put the boolean *True* (do you remember what a boolean is?). The boolean *True* is per definition always true.

You can also define a course of action in case the statement is false. This you can do with *else*:

```
if(False):  
    print("it is true")  
else:  
    print("it is false")
```

it is false

Comparison operators

In python you use the comparison operator `==` to decide if two elements are similar. Have look at this code:

```
varX = "ball"  
if (varX == "cube"):  
    print("good")  
else:  
    print("wrong")
```

wrong

Do you understand the difference between = and == ?

There are many more comparison operators:

<i>Code</i>	<i>Name</i>	<i>Description</i>	<i>Examples</i>
==	Equal	Returns True if two values are the same; returns False otherwise.	a == b
!=	Not Equal	True if two values are not the same.	a != b
>	Greater Than	True if the first number is greater than the second.	a > b
<	Less Than	True if the first number is less than the second.	a < b
>=	Greater Than or Equal To	True if the first number is greater than or equal to the second.	a >= b
<=	Less Than or Equal To	True if the first number is less than or equal to the second.	a <= b

Exercise 5.4

Demonstrate each comparison operator with Python. Herewith an example with the < (less than) operator:

```
varX = 113
if (varX < 114):
    print("varX is kleiner dan 114")
```

varX is kleiner dan 114

Maybe you still wonder why the statement *if (True):* is always true. Consider the code below:

```
varX = 113
print(type(varX < 114))
```

<class 'bool'>

Do you understand why it says: <class 'bool'> ?

Logical operators

A **logical operator** (or boolean operator) is used to evaluate a combination of comparisons. An example is:

```
x = 5
print((x > 3) and (x < 10))
```

True

As statement $(x > 3)$ is true and $(x < 10)$ is true. The result of $((x > 3) \text{ and } (x < 10))$ is also true.

If one of the statements is false then the complete statement is also false:

```
x = 7
print((x > 3) and (x < 5))
False
```

Other frequently used logical operators are: *or* and *not*:

Code	Description
and	Returns True when both values are True; returns False otherwise.
or	Returns True if either value is True; returns False if both values are False.
not	Returns True if the value is False; returns False if the value is True.

Book reference

- Read chapter 5: Scales, keys, and melodies
- Carry out the steps in Interlude 5: Lean on me.

Assignment 5

Have a look at the following video and do the exercise:

https://youtu.be/YSIIKe_2kG8

```
C = [48, 52, 55]
Am = [57, 60, 64]
F = [53, 57, 60]
G = [55, 59, 62]
E7 = [52, 56, 59, 62]

listHal = [C, Am, C, Am, F, G, C, G, C, F, G, Am, F, G, E7, Am]
durHal = [6, 6, 6, 6, 6, 6, 6, 6, 6, 3, 3, 6, 6, 6, 6, 6]
listChorus = [F, F, F, F, Am, Am, Am, Am, F, F, F, F, C, C, G, G, C, C]
for chord, duration in zip(listHal, durHal):
    for dur in range(0, duration):
        playNote(chord, beats=1)
```

Add a bassline and drums to the basic melody of Hallelujah.

You can get inspiration from here:

<https://www.hooktheory.com/theorytab/view/leonard-cohen/hallelujah>

Lesson 6: Diatonic chords and chord progression

Chapter 6 of the book is the most difficult up to now. Some new, abstract, concepts are introduced.

- On the side of musical theory you learn about what chords to combine and chord progression,
- with Python you learn about dictionaries (a new datatype), and
- you learn how to create your own modules with libraries in TunePad.

Do not worry if you do not understand everything. Particularly the theory about music is a complete study in itself.

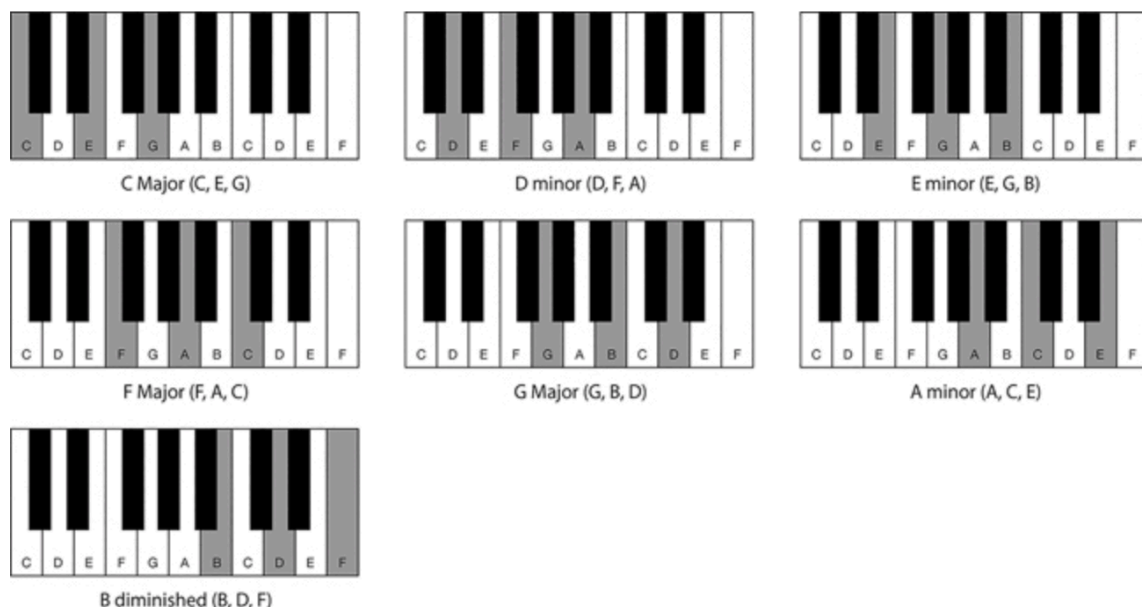
If you are short on time limit yourself to understanding the basics. However, in case you feel like it, dive into the subjects. If you thoroughly understand what is discussed, you are moving from the beginners phase to more advanced. Give it a try!

Chord progression

Many songs that are ‘pleasant’ to the ear contain notes and chords that are part of the dominant scale.

You already know that the C major scale only consists of ‘whole notes’ (the white keys of the piano).

The main other chords that you can make with the C major scale are the following:



A much used chord progression is to take the following chords:

- C major (I) – which is the first chord of the scale C .
- G major (V) – the fifth tone of the scale as root note.
- A minor (vi) – the sixth tone of the scale as root note.
- F major (VI) – the fourth note of the scale as root note.

Try the following code if you want to hear this progression of chords:

```
# Axis chord progression

I = [48, 52, 55] # C major chord [C, E, G]
V = [55, 59, 62] # G major chord [G, B, D]
vi = [57, 60, 64] # A minor chord [A, C, E]
IV = [53, 57, 60] # F major chord [F, A, C]

chords = [I, V, vi, IV]

for chord in chords:
    playNote(chord, beats=2)
```

You might notice the ‘weird’ names of the variables: I, V, vi, and IV. These are roman numerals to indicate the chord in relation to the root note of the scale. The roman numerals from 1 to 6 are I, II, III, IV, V, and VI. If written in capital letters they indicate a major chord (I, V, and IV). When using small letters they indicate a minor chord (vi).

This specific chord progression shifted from being famous to being infamous. Instrumental was the 4-chord sketch of the comedy group Axis of Awesome:
<https://youtu.be/5pidokakU4I>

Python intermezzo 6

You know already the following simple datatypes: *string*, *float*, *boolean* and *integer*. With datatype *list* you practiced with a more complex datatype.

In this intermezzo another datatype is introduced: **dictionary**.

A dictionary stores values in pairs and can be recognized through the ‘curly brackets’ { and }.

An example of a dictionary is:

```
dictCapitals = {"Netherlands": "Amsterdam", "France": "Paris", "Italy": "Rome"}
```

In the above example the countries (such as “Netherlands”) are the keys and the capitals (“Amsterdam”) the value.

Underneath you can see how such key:value pairs are used:

```
dictCapitals = {"Netherlands": "Amsterdam", "France": "Paris", "Italy": "Rome"}
print(dictCapitals["France"])

Paris
```

So, how is it possible to find the country of a certain capital? You could try this out.

```
dictCapitals = {"Netherlands":"Amsterdam", "France":"Paris", "Italy":"Rome"}
print(dictCapitals["Rome"])
```

```
-----
KeyError                                Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 dictCapitals = {"Netherlands":"Amsterdam", "France":"Paris", "Italy":"R
ome"}
----> 2 print(dictCapitals["Rome"])

KeyError: 'Rome'
```

As you get an error, you know this is not the right way to find the key of a value. You might want to solve this by looping through the dictionary. To start try to find out how a loop works.

```
dictCapitals = {"Netherlands":"Amsterdam", "France":"Paris", "Italy":"Rome"}

for i in dictCapitals:
    print(i)
```

```
Netherlands
France
Italy
```

So, now how to find the capitals of each county:

```
dictCapitals = {"Netherlands":"Amsterdam", "France":"Paris", "Italy":"Rome"}

for i in dictCapitals:
    print(dictCapitals[i])
```

```
Amsterdam
Paris
Rome
```

You can find the capital of a specific country (such as “France”) with an if-statement:

```
dictCapitals = {"Netherlands":"Amsterdam", "France":"Paris", "Italy":"Rome"}

for i in dictCapitals:
    if (dictCapitals[i] == "Paris"):
        print("The capital of " + i + " is " + dictCapitals[i])
```

```
The capital of France is Paris
```




- Read chapter 6: Diatonic chords and chord progressions.
- Carry out the steps in Interlude 6: Random chord progressions

Create a program that import the module `chordCreation` and uses the `buildChords()` function to create the 'Axis'-chord progression I, V, vi, VI.

Final Exercise week 1

Swing is a style of jazz music that became popular in the 1930s. It has an up tempo music with a strong rhythm section (standup bass and drums). It usually has 4 beats per bar with an emphasis on the offbeat (this is on count 2 and 4).

More on swing:

<https://youtu.be/bGiPJZ-wRb4>

The musician Wynton Marsalis on swing:

https://youtu.be/_mLvytV2GrA

In this final exercise for the first week you are going to code a variation of the song “It Don't Mean a Thing (If It Ain't Got That Swing)” composed by Duke Ellington in 1931.

<https://youtu.be/qDQpZT3GhDg>

The song has been covered numerous times. Interesting is the video of the cooperation between the band Zuco103 and Joe Jackson:

<https://youtu.be/UVG92LRMZ2Y>

The assignment

You have to make your own interpretation of the song. This should be exactly 12 bars of 4 beats.

The bpm should be 160

The scale used should be: G, A, B \flat , C, D, E \flat , and F.

B \flat is the black key between A and B (on the piano). The D \flat is the black key between C and E \flat . E \flat is the black key between D and E.

The equivalent of [G, A, B \flat , C, D \flat , E \flat , F] in MIDI can be [55, 57, 58, 60, 61, 63, 65]

What do you do:

- Create a drum part with emphasis on the 2nd and 3rd beat
- Create an easy bass part consisting of the notes in the scale
- Add a melody (see the melodies below)
- It should be exactly 12 bars of 4 beats. Of course you can use loops
- Add whatever you find is interesting.



The screenshot shows a music notation software interface. At the top, there is a control bar with a play button, a stop button, and a next button. Below this is a piano roll with a timeline from BEAT 2 to 16. The piano roll displays two horizontal bars representing notes: one at a lower pitch (blue) and one at a higher pitch (light blue). The code editor below the piano roll contains the following code:

```

1 # it don't mean a thing
2 playNote(55, 2)
3 playNote(55)
4 playNote(58)
5
6 playNote(61, 2)
7 playNote(55)
8 playNote(58)
9
10 playNote(61, 2)
11 playNote(60, 0.5)
12 playNote(58)
13 playNote(55, 2.5)
14 rest.(2.).

```

The screenshot shows a code editor with the following code:

```

1 # New TunePad cell
2 playNote(55, 2)
3 playNote(55)
4 playNote(58)
5
6 playNote(61, 2.5)
7 playNote(58)
8 playNote(55, 0.5)
9
10 playNote(58)
11 playNote(58)
12 playNote(58)
13 playNote(55)
14
15 playNote(58, 2)
16 rest(1)
17 playNote.(61.).

```

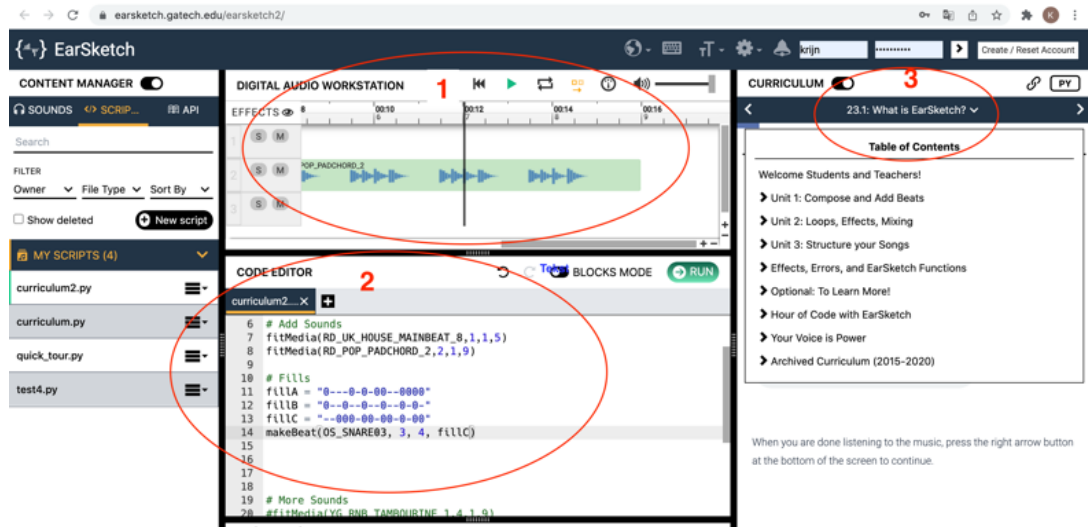
les 7: EarSketch intro

Another option to code music with Python is the online environment EarSketch:

<https://ears sketch.gatech.edu/ears sketch2/>

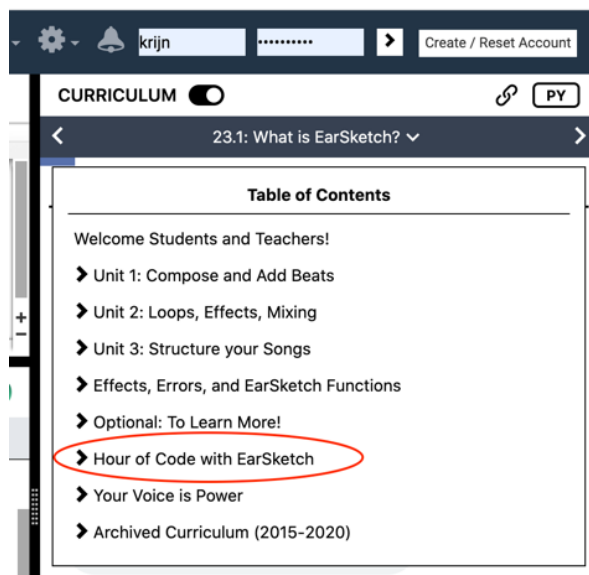
Take the following steps to start out with EarSketch:

- 1) Go to the online environment and create an account. You will be directed to the following screen:

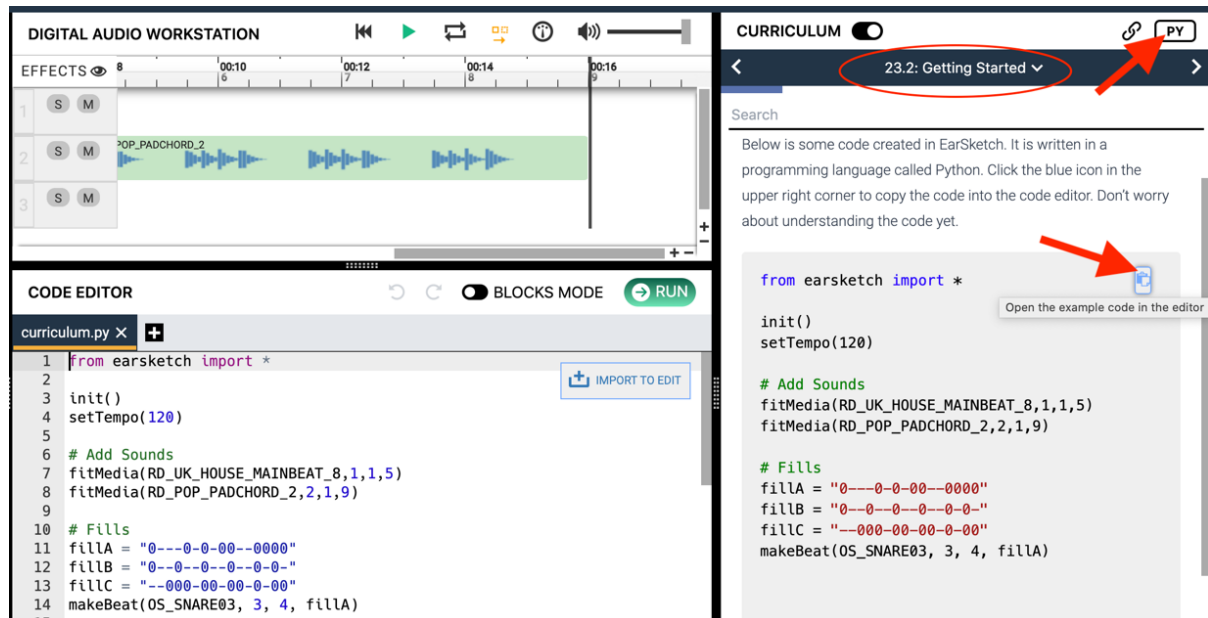


- On 1 you see the **Digital Audio Workstation (DAW)**. This shows which music is being played.
- On 2 is the Python editor to create the music with.
- On 3 it shows the online curriculum. You will start with the 'Hour of Code'.

- 2) Choose at the right site (Curriculum) for 'Hour of Code with EarSketch':



- 2) At 'Hour of Code with EarSketch' go to section 23.3. Ensure that it says 'PY' on the right side on top (PY stands for Python). It should not be 'JS' (this will bring you to the JavaScript editor, another programming language).
- 3) Click on the icon on the right '*from earsketch import **' (see image). The Python code in the editor in the middle of the screen will appear.



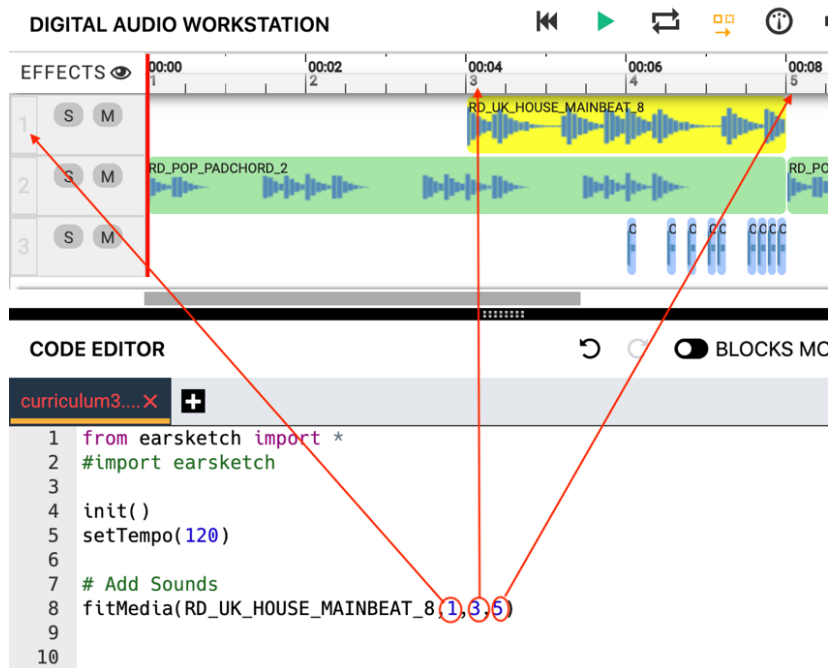
Take a moment to have a closer look at the Python code (see image below):

```

1  from earsketch import *
2
3  init()
4  setTempo(120)
5
6  # Add Sounds
7  fitMedia(RD_UK_HOUSE_MAINBEAT_8,1,1,5)
8  fitMedia(RD_POP_PADCHORD_2,2,1,9)
9
10 # Fills
11 fillA = "0---0-0-00--0000"
12 fillB = "0--0--0--0--0-0-"
13 fillC = "--000-00-00-0-00"
14 makeBeat(OS_SNARE03, 3, 4, fillA)
15
16 # More Sounds
17 #fitMedia(YG_RNB_TAMBOURINE_1,4,1,9)
18
19 finish()
```

- All lines with # are comments (lines 6, 10, 16 and 17). As you remember from the previous lessons; Python ignores them.
- The line 1, *from earsketch import ** ensures the import of the EarSketch library and the specific functionalities.
- *Init()* in line 3, is a mandatory EarSketch function to start the Digital Audio Workstation (DAW).
- The EarSketch *setTempo(120)* function manages the beats per minute (bpm).

- In the lines with `fitMedia()` (7 and 8) an audio file is added to the track. For example in `fitMedia(RD_UK_HOUSE_MAINBEAT_8,1,3,5)` the following happens:
 - o Audio track RD_UK_HOUSE_MAINBEAT_8 is added.
 - o The audio appears at beat 3 and ends at beat 5 in the DAW.
 - o This audio becomes track 1 (see image).



- Lines 11, 12 and 13 contain the variables `fillA`, `fillB`, and `fillC`. These variables have datatype string. Such strings (containing - and 0's) can be used to create beats with the `makeBeat()` function. The dash - signifies a rest, an 0 signifies that the sample is played.. In this case they are 16 characters (4 bars of 4).

```

1  from earsketch import *
2
3  init()
4  setTempo(120)
5
6  # Add Sounds
7  fitMedia(RD_UK_HOUSE_MAINBEAT_8,1,1,5)
8  fitMedia(RD_POP_PADCHORD_2,2,1,9)
9
10 # Fills
11 fillA = "0---0-0-00--0000"
12 fillB = "0--0--0--0--0-0-"
13 fillC = "--000-00-00-0-00"
14 makeBeat(OS_SNARE03, 3, 4, fillA)
15
16 # More Sounds
17 #fitMedia(YG_RNB_TAMBOURINE_1,4,1,9)
18
19 finish()

```

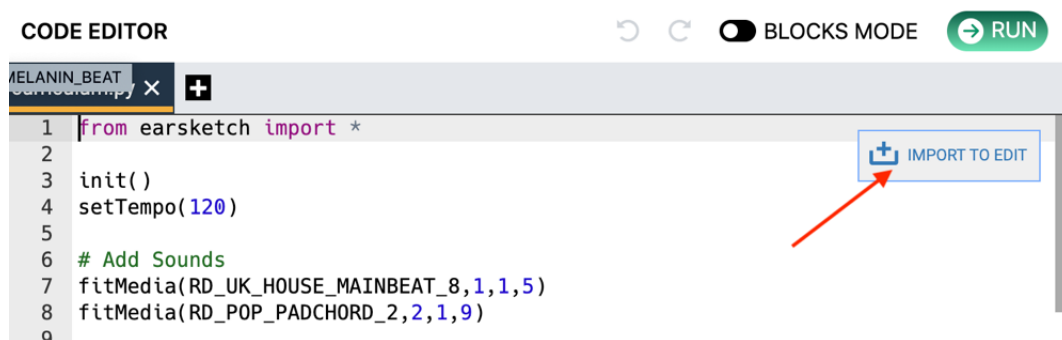
Exercise 7.1

Answer the following questions regarding function *makeBeat(OS_SNARE03, 3, 4, fillA)*:

- a) Which sound is used?
- b) What track in the DAW is used for this sound?
- c) What is start beat and end beat of the sound?
- d) How many times the sample is played in this string (and how did you find it)?

Exercise 7.2

Change *setTempo* from 120 to 100. In order to do this click on ‘import to edit’ and give the file a different name. Now you can change the function *setTempo()*.



Python intermezzo 7

In the previous chapters you learned already many Python concepts in TunePad. You will notice that the same concepts are used in EarSketch.

Likely you will be able to answer the following questions on Python. If not, please read the previous Python intermezzo's again.

Exercise 7.3

- a) How can you add comments in Python?
- b) What is a library?
- c) How do you add a library to your code?
- d) What is a function?
- e) Can you name a function that is embedded in EarSketch?

In EarSketch contains a Python environment to create music from samples.

However, it also has ‘standard’ Python functionality, e.g. you can print output on the screen (previously you might have used Jupyter for this).

In the code underneath the variable *varX* is created with the value “Hallo Wereld!” (Hello World in Dutch). After that the value is printed on the output part:

```

CODE EDITOR
basics.py x +
1 from earsketch import *
2
3 init()
4 setTempo(120)
5
6 varX = "Hallo Wereld!"
7 print(varX)
8
9 finish()

Running script...
Hallo Wereld!
Script ran successfully

```

Variables can be used by creating music. You have done this with TunePad and again now with EarSketch.

In the code underneath fillA, fillB and fillC are variables. fillA is used in the function makeBeat().

```

CODE EDITOR
basics.py x basics2.py x +
1 from earsketch import *
2 init()
3 setTempo(120)
4
5 fitMedia(RD_UK_HOUSE_MAINBEAT_8,1,1,5)
6 fitMedia(RD_POP_PADCHORD_2,2,1,9)
7
8 fillA = "0---0-0-00--0000"
9 fillB = "0--0--0--0--0-0-"
10 fillC = "--000-00-00-0-00"
11 makeBeat(OS_SNARE03, 3, 4, fillA)
12
13 finish()
14

```

Exercise 7.4

How can you print the value of fillA on the screen?

Datatypes lists and strings.

You know that:

- A string is a sequence of characters such as "00_0_"
- A list is a sequence of objects such as [0, 0, _, 0, _]

There is a similarity between the two datatypes. The difference is that a string always contains characters and a list can also contain other datatypes such as integers or booleans.

In EarSketch you have seen that a string can be used to create a sequence of sounds:

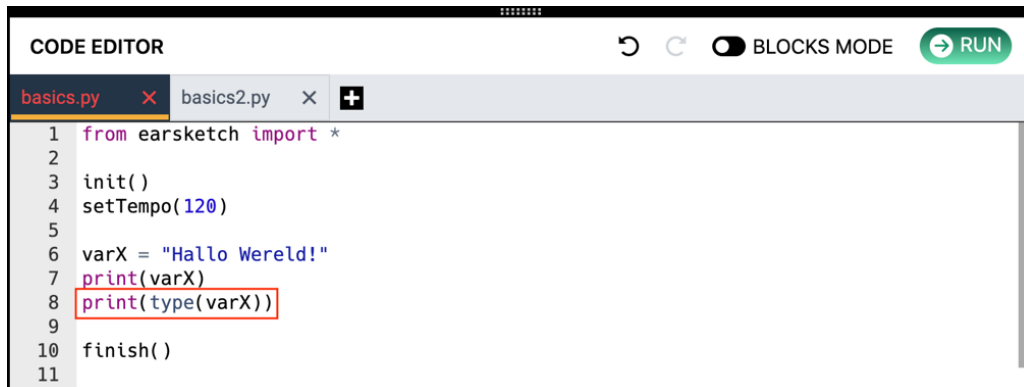
fillA = "0- - -0-0-00- -0000"

In NotePad you have seen that a list can be used to create a sequence of sounds

for i in [1, _, _, 1, _, 1, _, 1, 1, _, _, 1, 1, 1, 1]:
playNote(i)
rest(0.25)

Very often there are different solutions possible to get to the same result (such as the creation of a sequence of sounds). The *art of programming* is the choices you make (and just as in other arts: sometimes the choice depends on your own preference).

The function `type()` can be used to find the datatype of a variable. Try out the code underneath. What is the result of `print(type(varX))`?



```
CODE EDITOR
basics.py basics2.py
1 from earsketch import *
2
3 init()
4 setTempo(120)
5
6 varX = "Hallo Wereld!"
7 print(varX)
8 print(type(varX))
9
10 finish()
11
```

Changing datatypes

Sometimes it can be useful if the datatype is changed from one type to another. Of course that is not always possible. The string “Apple” does not have an equivalent decimal number. However, the string “3” can be changed into an integer 3 with the following structure

To change a string into an integer:

```
varStr = "3"
varInt = int(varStr)
print(varInt)
print(type(varInt))
```

```
3
<class 'int'>
```

To change an integer into a string:

```
varInt = 5
varStr = str(varInt)
print(varStr)
print(type(varStr))
```

```
5
<class 'str'>
```

To change a string into a list:

```
varString = "test"
makeList = list(varString)
print(makeList)
```

```
['t', 'e', 's', 't']
```

To change a list into a string:

```
varList= [1,0,1,0,0,1]
varString = str(varList)
print(varString)
print(type(varString))

[1, 0, 1, 0, 0, 1]
<class 'str'>
```

Datatypes: strings, integers, floats and boolean

In the previous chapters, you have learned already about the simple datatypes int, string, float, and boolean. As repetition is the key to learning, herewith some more exercises on simple datatypes:

Variables of the datatype int (integer) are complete numbers such as 9, 204, and -3
Strings are sequences of characters. Try out the following code:

```
varA = 8
varB = varA + varA
print(varB)
```

```
varC = "8"
varD = varC + varC
print(varD)
```

```
varE = 8
varF = "8"
varG = varE+varF
print(varG)
```

Explain the difference in results of the print() statements above.

A **float** is een getal met 1 of meerdere decimalen (bijvoorbeeld 1,10 of 0,2).

A **boolean** can only be one of two values, either true or false (1 or 0). A boolean can be used in evaluating whether a certain statement is true (or false).

In the code underneath two variables are added and it is verified whether the results is larger than 10. Do you see how a boolean is used?

```
varX = 6
varY = 7
varZ = varY + varX
print(varZ==13)
print(type(varZ == 13))
```

Exercise 7.6

What do you see in the output part of EarSketch when executing the following code?

```
varBool = True
print(type(varBool))

varInt = 1300
print(type(varInt))

varFloat = 142.242
print(type(varFloat))

varString = "hello"
print(type(varString))
```

Book reference

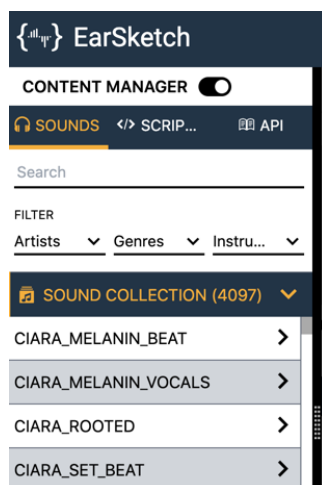
- Read chapter 9: Song composition and EarSketch
- Carry out steps in Interlude 8: How to make a drum fill.

Assignment lesson 7

This assignment consists of 3 parts.

Create an EarSketch track of at least two bars that has a beat sequence. You can use the `makeBeat()` function and be inspired by `fillA`, `fillB` and `fillC` (of the examples above).

Add another track to the composition. Choose a sound from the sound library (left side of the screen) and use this sound with a `fitMedia()` function.



The two tracks should go together.
Of course you can add more tracks.

lesson 8: EarSketch remix

In this chapter you are going to remix the existing song *Entrepreneur* by Pharrell Williams and Jay Z. EarSketch cooperates with Young Guru, the sound engineer of Jay Z. The EarSketch content manager contains many samples of *Entrepreneur*.

Take some time to enjoy the song:

<https://youtu.be/bTOoY5MIkvM>

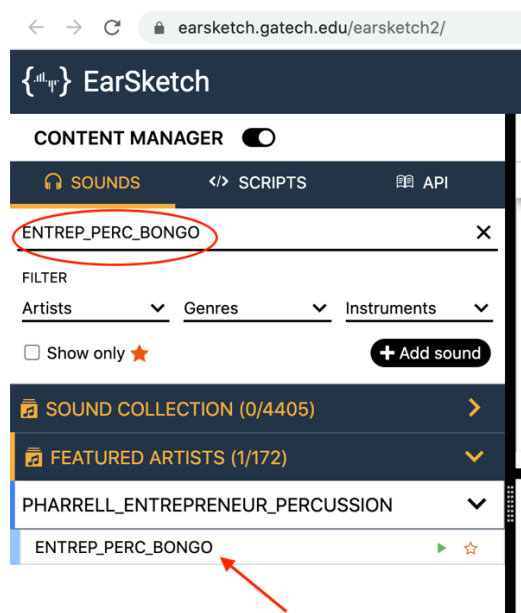
The song is meant as an ode to black entrepreneurs but can also be inspiring for anyone who wants to change something in society or their own circumstances.

*You gotta let go (Let go)
If you want to fly, take the leap
You gotta risk it all (Risk it all)
Or they'll be lots of things you'll never see
You gotta let, let go
'Cause you never know what's in store
Mister entrepreneur*

You will make a layered remix (using different tracks) of *Entrepreneur*. You will use the following samples:

- Drums → ENTREP_PERC_BONGO
- Vox → ENTREP_VOX_BK_EHH
- Vox1 → ENTREP_VOX_BK_LETOUT
- Orch → ENTREP_THEME_ORCH
- Bass → ENTREP_THEME_BASS_1
- Keys → ENTREP_THEME_KEYS_2

The samples you can find by using the search in the content-manager:



To create a new composition you have to open a new file. This you can do by clicking on the + . Do not forget to give the file a nice and meaningful name.

CODE EDITOR

curriculum4.... X remix.py X +

```
10 setTempo(90)
11 #music
12 #SOUNDBANK
13 drums=ENTREP_PERC_BONGO
14 vox=ENTREP_VOX_BK_EHH
```

The next step is to add the first layer of music. For example the drum sample ENTREP_PERC_BONGO . Because this name is quite long we create the variable drums (see image).

Because it is a hip hop song we lower the tempo a bit to 90.

```
1 from earsketch import *
2
3 init()
4 setTempo(90)
5
6 drums=ENTREP_PERC_BONGO
7
8 fitMedia(drums, 1, 1, 13)
9
10 finish()
11
```

Listen to the result. Do you like it?

Now its time to add the second layer (this we do in the same file). We choose a voice sample:

```
1 from earsketch import *
2
3 init()
4 setTempo(90)
5
6 drums=ENTREP_PERC_BONGO
7 vox=ENTREP_VOX_BK_EHH
8
9 fitMedia(drums, 1, 1, 13)
10 fitMedia(vox,2, 2, 12)
11
12 finish()
```

How does it sound? Are you happy, or do you want to change something?

The next step is to add a bass sample:

```
1 from earsketch import *
2
3 init()
4 setTempo(90)
5
6 drums=ENTREP_PERC_BONGO
7 vox=ENTREP_VOX_BK_EHH
8 bass=ENTREP_THEME_BASS_1
9
10 fitMedia(drums, 1, 1, 13)
11 fitMedia(vox, 2, 2, 12)
12 fitMedia(bass, 3, 3, 11)
13
14 finish()
```

Exercise 8.1

Add the following samples in a creative way to your remix:

```
vox1=ENTREP_VOX_BK_LETOUT
orch=ENTREP_THEME_ORCH
keys=ENTREP_THEME_KEYS_2
```

Python intermezzo 8

In the previous lessons you have worked with loops in different ways. Not only did the NotePad and EarSketch environments have built-in loops, also you know how to work with a for loop.

While loop

A for loop is used when we are aware of the number iterations at the time of execution. For example in the code underneath we know in advance that there will be four iterations:

```
for fruit in ["apple", "pear", "banana", "orange"]:
    print(fruit)
```

```
apple
pear
banana
orange
```

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

However, we can also use loops when we do not know in advance how many iterations they are. In these cases we can use a while loop. In the example below we

print a random integer between 0 and 3 as long as the random integer does not become 2. The comparison (*varRand* != 2) is *True* as long as *varRand* does not (!=) have the value 2. Do you realise that (*varRand* != 2) is a boolean?

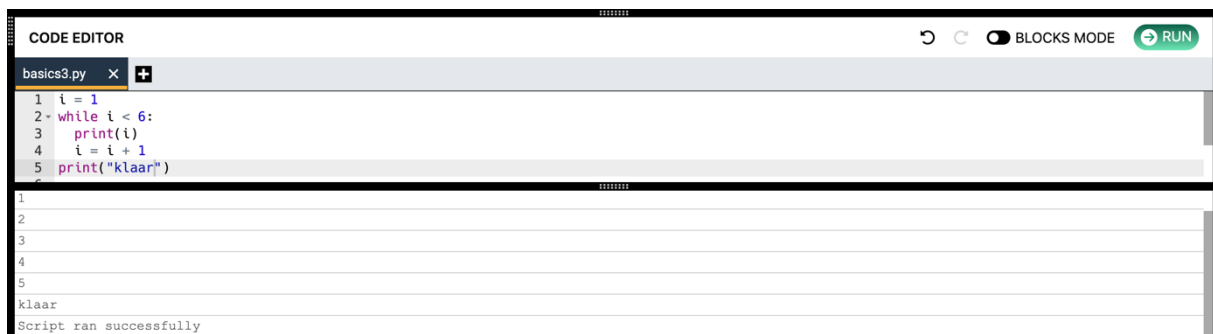
```
import random
varRand = 0
while (varRand != 2):
    print(varRand)
    varRand = random.randint(0,3)
```

```
0
0
1
0
3
```

```
import random
varRand = 0
while (varRand != 2):
    print(varRand)
    varRand = random.randint(0,3)
```

```
0
1
3
```

Herewith another while loop example. In this case you could also have used a for-loop.

A screenshot of a web-based code editor interface. The top bar shows 'CODE EDITOR' on the left, and navigation icons (back, forward, refresh) and a 'BLOCKS MODE' toggle on the right. Below the bar, a tab labeled 'basics3.py' is active. The code editor contains five lines of Python code: 1. i = 1, 2. while i < 6:, 3. print(i), 4. i = i + 1, 5. print("klaar"). The output area below the code shows the numbers 1, 2, 3, 4, 5 on separate lines, followed by the word 'klaar' on the sixth line. At the bottom, a status message reads 'Script ran successfully'.

On line 1 the variable *i* gets value 1.

On line 2 the while loop starts

As long as *i* is smaller than 6 (*i* < 6) the indented lines (3 and 4) are executed.

Exercise 8.2

Change the code so you get the output underneath (*klaar* means ready in Dutch).

```
Running script...
1
klaar
2
klaar
3
klaar
4
klaar
5
```

Also in this example does `while i < 6` the indented actions as long as `i < 6` is *True* is.

The result `i < 6` is a boolean (True of False).

Try out this code. Do you understand the output?

```
CODE EDITOR
basics3.py x +
1 i = 1
2 while i < 6:
3     print(type(i<6))
4     print(i<6)
5     i = i + 1
6     print("klaar")
```

Loops in music

Loops are very common in music. In the previous lesson you have worked with the code underneath. There was a loop from sample `RD_UK_HOUSE_MAINBEAT_8` from beat 1 until 5.

The sample `RD_POP_PADCHORD_2` was repeated from 1 to 9.

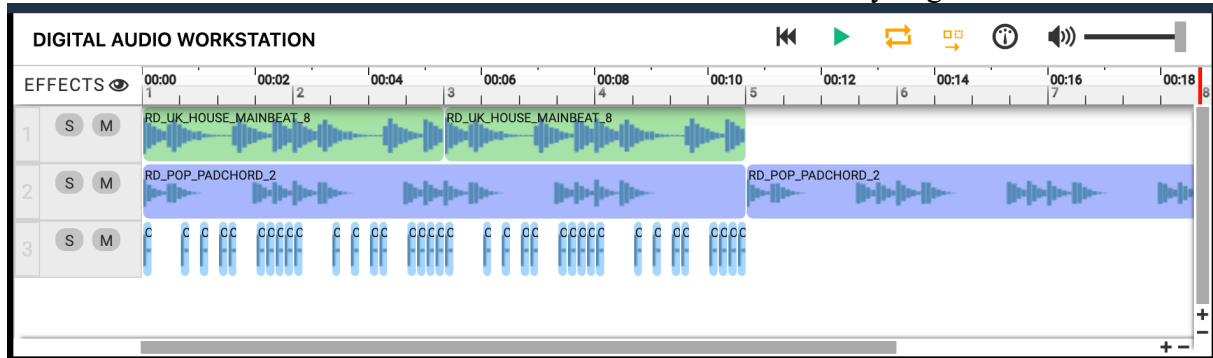
However, `makeBeat(O_SNARE03, 3, 4, fillA)` had no repetition. The sequence `fillA` with rhythm `0---0-0-00---0000` was carried out once.

```
CODE EDITOR
curriculum.py x +
1 from earsketch import *
2
3 init()
4 setTempo(90)
5
6 # Add Sounds
7 fitMedia(RD_UK_HOUSE_MAINBEAT_8, 1, 1, 5)
8 fitMedia(RD_POP_PADCHORD_2, 2, 1, 9)
9
10 # Fills
11 fillA = "0---0-0-00---0000"
12 makeBeat(OS_SNARE03, 3, 4, fillA)
13
14 finish()
```


The function `makeBeat()` does not have an built-in functionality to arrange repetition. This can be done with regular Python loops.

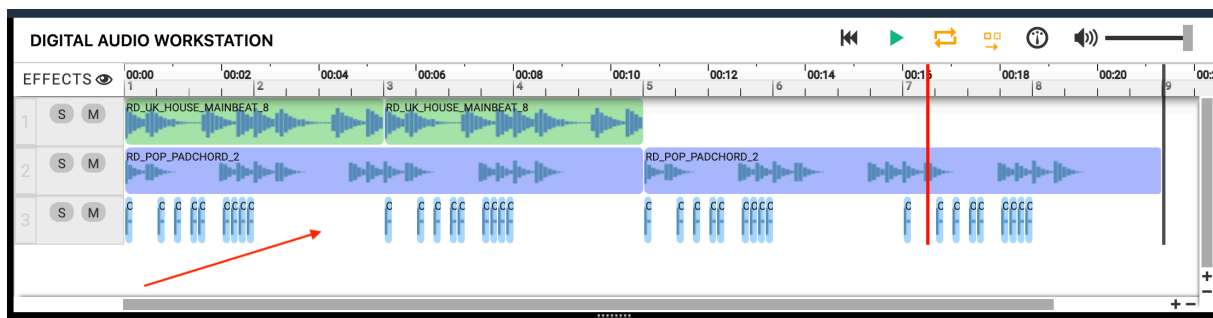
In the code underneath the beat sequence is carried out 5 times (just as long as the sample `RD_UK_HOUSE_MAINBEAT_8`).

The DAW now shows the result underneath. Is this similar to what you get?



Exercise 8.3

What should you change to get the result underneath?



Book reference

- Carry out steps in Interlude 9: How to make a snare drum riser.

Assignment 8

Remix with loops

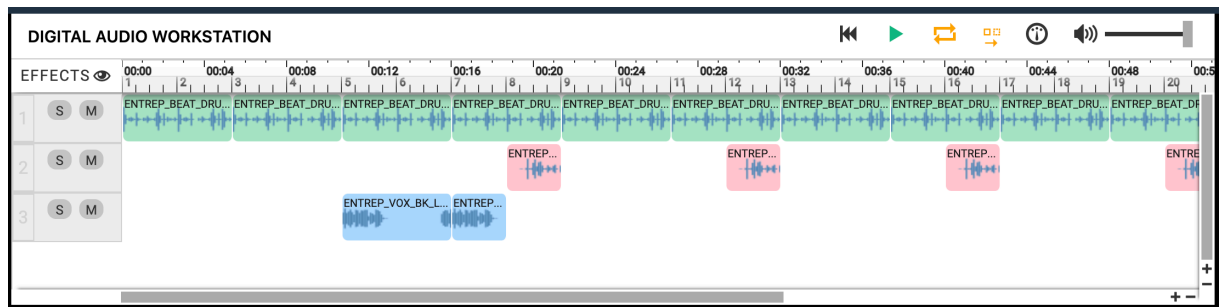
You will continue to work on the remix of the song *Entrepreneur* by Pharrell Williams and Jay Z.

Listen again to the song:

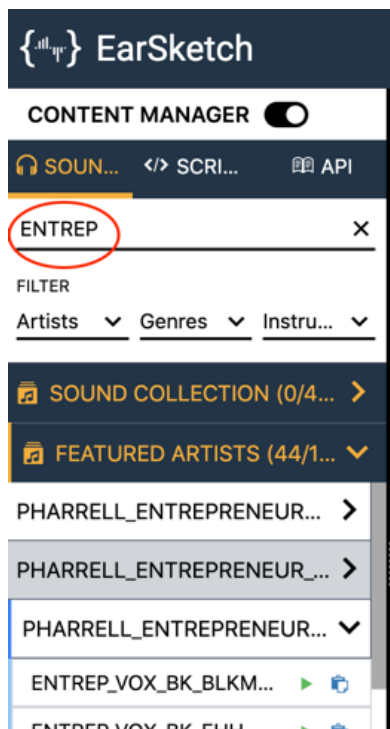
<https://youtu.be/bTOoY5MIkvM>

The remix must be around 30 seconds and contain the following elements:

- 1) The samples ENTREP_BEAT_DRUMBEAT, ENTREP_PERC_WHISTLE and ENTREP_VOX_BK_LETOUT in the following way:



- 2) Two other fitMedia() constructs with sounds from Entrepreneur (search on ENTREP in the Content Manager):



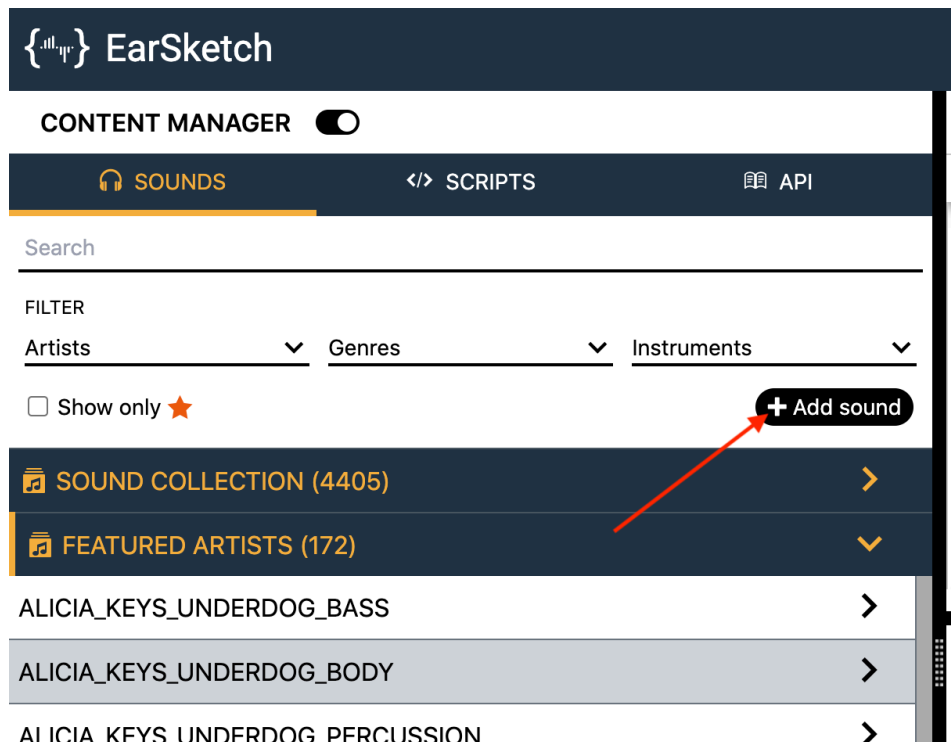
- 3) A OS_SNARE06 makeBeat() with the following sequence "0-0---0-0---"
- 4) At least two other samples of Entrepreneur.

Final assignment week 2

For your final assignment you create your own composition. You can do this alone or in a small group of 2 or 3. The composition is at least 90 seconds

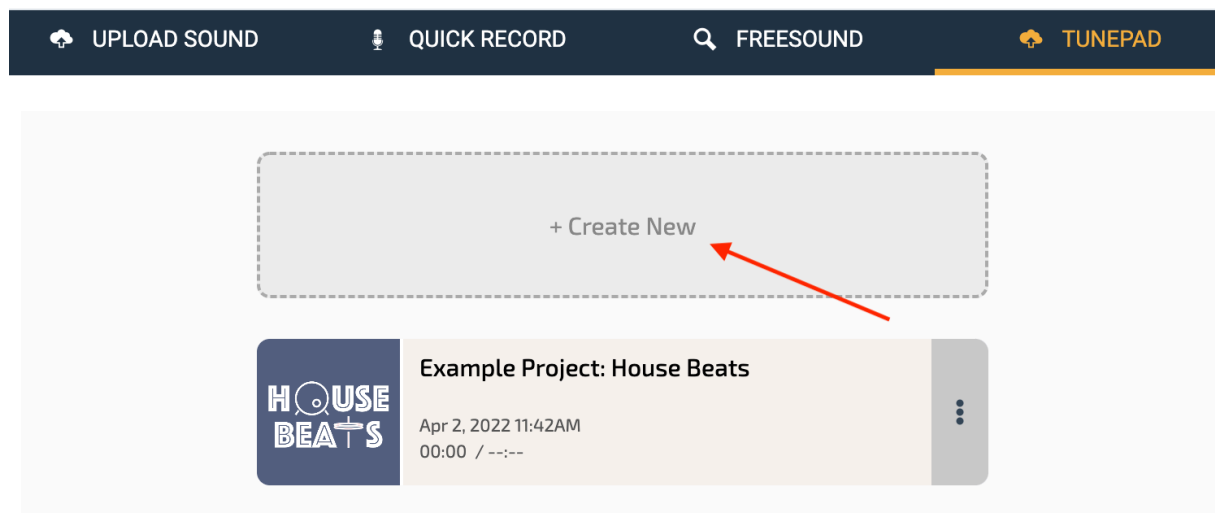
You need to use at least two samples of which one was created by TunePad. Have a look at the images underneath on how to import samples from TunePad.

Step 1) Click on '+ Add sound' at the EarSketch content manager




Step 2) Go to the 'TUNEPAD' tab and click on create new.


Add a New Sound





If you want to add an existing sample to the content manager you can click on ‘UPLOAD SOUND’ and choose a file in the correct format.

Add a New Sound

 **UPLOAD SOUND**

 **QUICK RECORD**

 **FREESOUND**

 **TUNEPAD**

 Choose a file...

.wav

.aiff

.mp3

Constant Name (required)

KRIJN_VOX

Tempo (Optional)

120

CANCEL

UPLOAD

If you have a hard time getting inspiration for this project the advice is to just start:

Some inspiration on how to create songs:

Stromae:

<https://youtu.be/G4lTg4Xq5AM>

create song by Ben Folds:

https://youtu.be/BytUY_AwTUs

