COMP1021
Introduction to Computer Science

Using the Slice Notation

David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Use the slice notation to get a certain part of items from a list

The Slice Notation

- This presentation discusses the *Slice Notation*
- The slice notation is a set of numbers, separated by colons and put between square brackets after the name of a list or a tuple:

`name_of_list_or_tuple[` *Start* **:** *End* **:** *Step* `]`

The Meaning of the Numbers

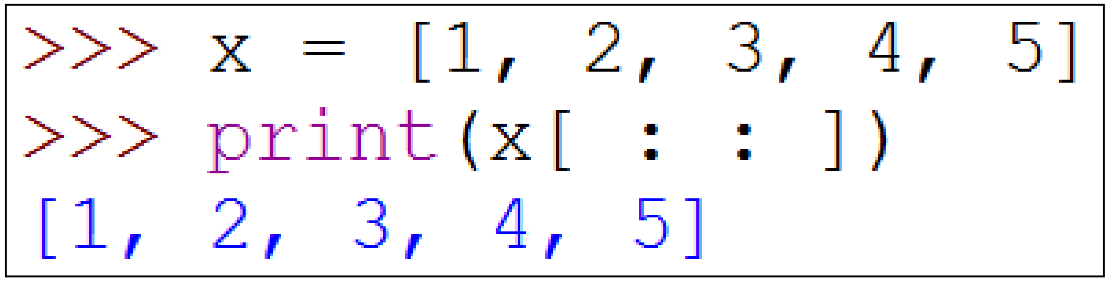
- The three numbers in the slice notation have very similar meaning to the numbers used by `range()`

`name_of_list_or_tuple[` *Start* : *End* : *Step* `]`

Start	extract items starting from this index
End	extract items up to **and not including** this index
Step	increase the item index using this step value, i.e. skipping items

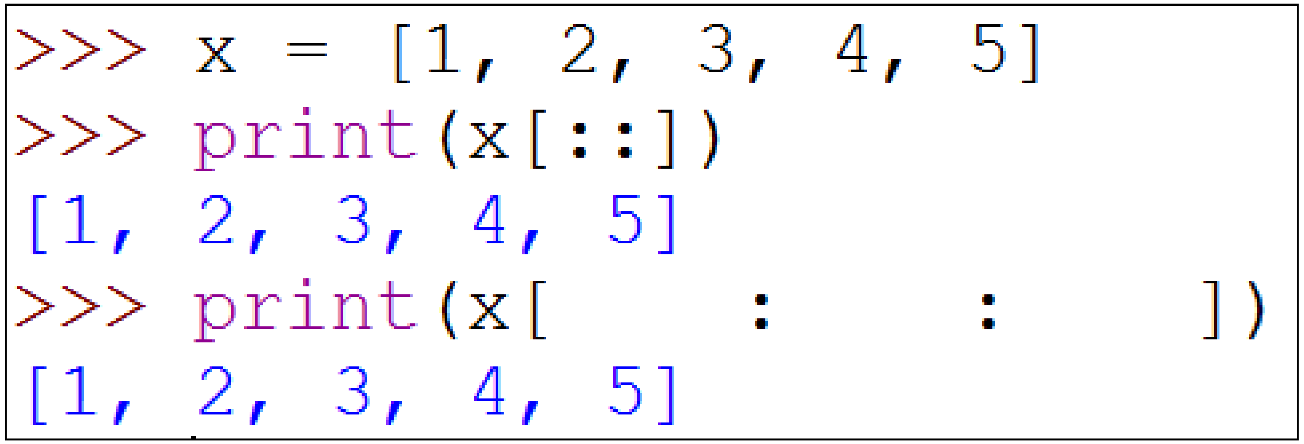
A Simple Example

- If you don’t write the numbers, Python automatically uses appropriate values:



```
>>> x = [1, 2, 3, 4, 5]
>>> print(x[ : : ])
[1, 2, 3, 4, 5]
```

- Spaces aren’t important:



```
>>> x = [1, 2, 3, 4, 5]
>>> print(x[::])
[1, 2, 3, 4, 5]
>>> print(x[      :      :      ])
[1, 2, 3, 4, 5]
```

More Examples

- Let's assume we have a list `x`, which looks like this:

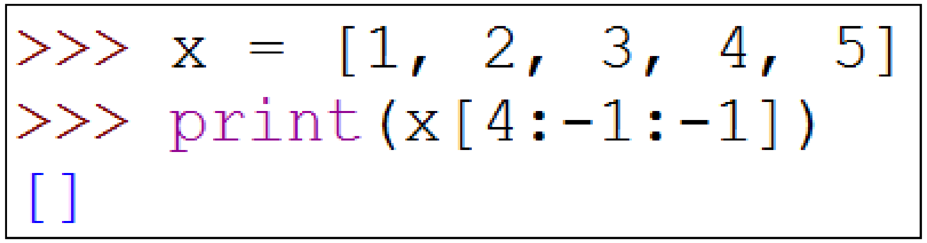
$$\texttt{x = }\underbrace{\texttt{[1,}}_{0}\ \underbrace{\texttt{2,}}_{1}\ \underbrace{\texttt{3,}}_{2}\ \underbrace{\texttt{4,}}_{3}\ \underbrace{\texttt{5]}}_{4}$$

- Here are some examples of slicing:
  - `x[0:3]` returns `[1, 2, 3]`
  - `x[0:5:2]` returns `[1, 3, 5]`
  - `x[3:]` returns `[4, 5]`
  - `x[:3]` returns `[1, 2, 3]`
  - `x[4:0:-1]` returns $\underline{\texttt{[5, 4, 3, 2]}}$

Where is the first item?

Reversing a List Using Slicing 1/2

- You have seen this example in the previous page:
  - `x[4:0:-1]` returns `[5, 4, 3, 2]`
- The first item of the list is not included in the above example because the end number is 0
- You may then think that `x[4:-1:-1]` will give you `[5, 4, 3, 2, 1]`
- However, it won't work because **negative indices have a special meaning**



```
>>> x = [1, 2, 3, 4, 5]
>>> print(x[4:-1:-1])
[]
```

An empty list i.e. no result

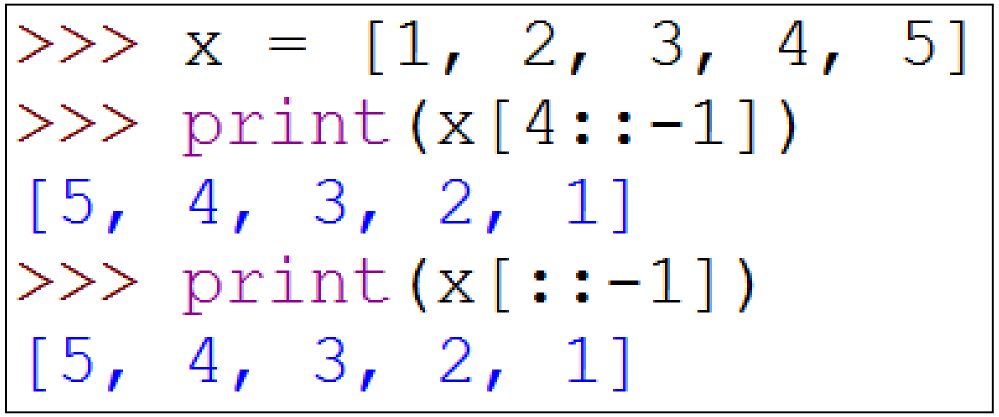
Reversing a List Using Slicing 2/2

- Instead of using `-1` as the end number you can return the reversed list of `x` like this:

```
x[4::-1]
```

or simply omit both the start and end numbers:

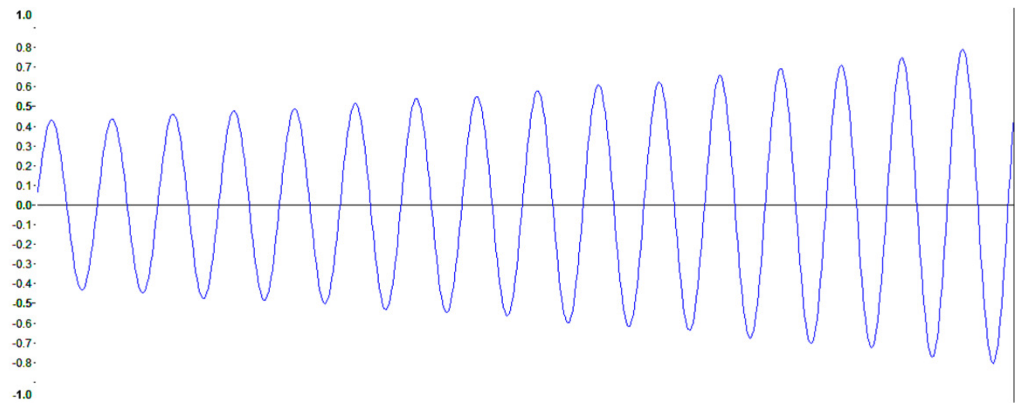
```
x[::-1]
```



```
>>> x = [1, 2, 3, 4, 5]
>>> print(x[4::-1])
[5, 4, 3, 2, 1]
>>> print(x[::-1])
[5, 4, 3, 2, 1]
```

Digital Audio

- A sound file consists of a sequence of values, called audio *samples* (a sample is simply a number)
- These audio samples can be positive or negative
- The sequence of values forms the shape of the sound wave, which represents the sound



Some audio samples, shown in audio editing software

Time

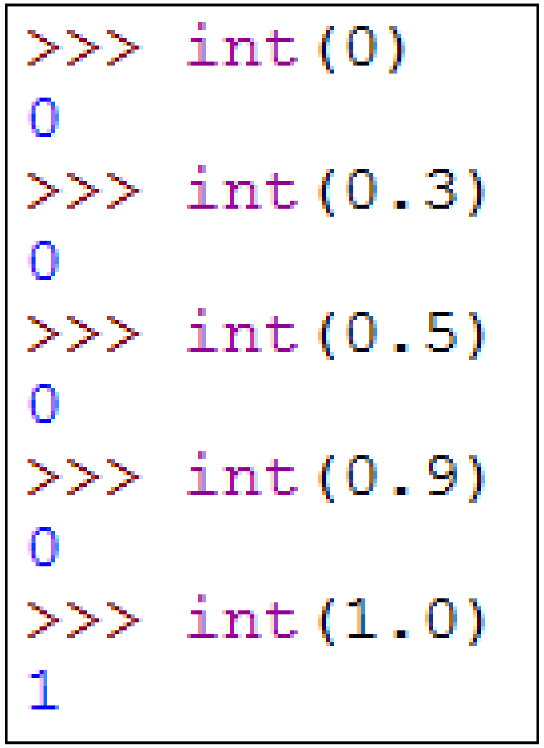
Accessing Precise Sections

- Digital audio uses a fixed number of samples for each second
- In the COMP1021 WAV files, 44100 samples are used for every second of audio

```
# Access the first second of the audio
samples[:44100]
# Access the third second of the audio
samples[44100*2:44100*3]
# Access the third second of the audio backwards
samples[44100*3:44100*2:-1]
```

Converting from a Float to an Integer Number

- A *float* is a number with a decimal place i.e. 3.1415
- We need to convert a float to an integer in the following examples, when we refer to items in a list
- `int()` converts a float to an integer
- It simply 'throws away' the decimal place (no rounding)



```
>>> int(0)
0
>>> int(0.3)
0
>>> int(0.5)
0
>>> int(0.9)
0
>>> int(1.0)
1
```

Accessing General Sections

```
# The first half of the audio
samples[ :int(len(samples)/2)]
# The first 25% of the audio
samples[ :int(len(samples)*.25)]
```

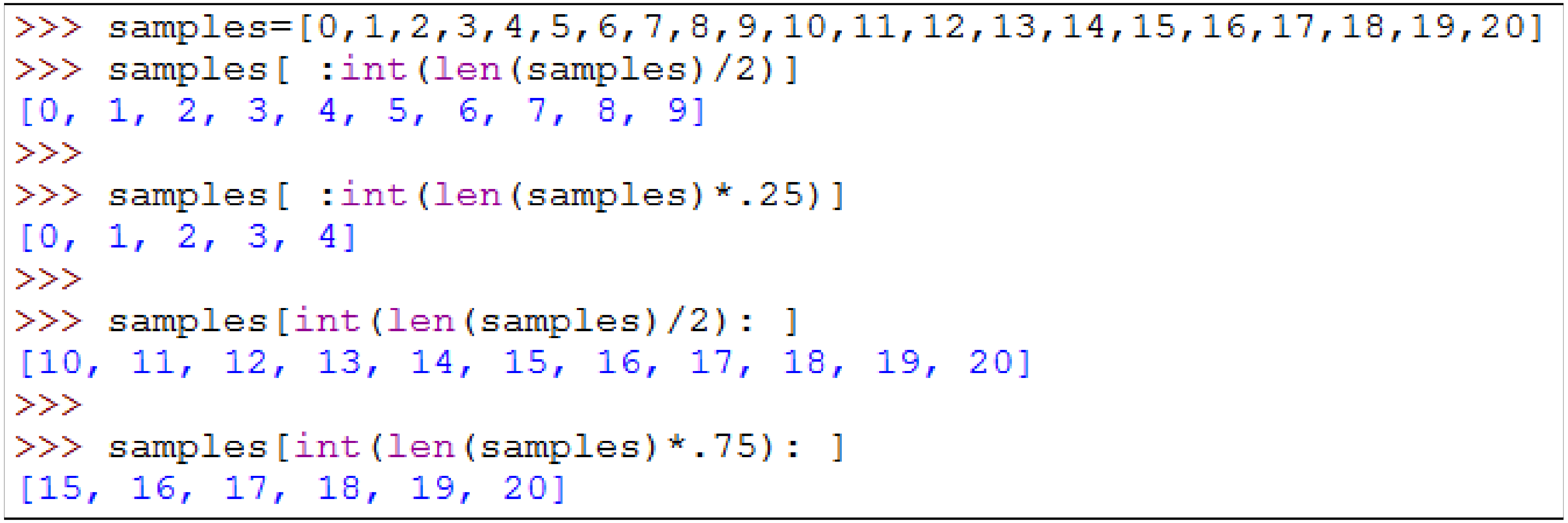
No start number is given, so Python will start at the beginning

`len()` *returns the number of items in a list*

```
# The last half of the audio
samples[int(len(samples)/2): ]
# The last 25% of the audio
samples[int(len(samples)*.75): ]
```

No end number is given, so Python will stop at the end

Trying it in the Shell



```
>>> samples=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
>>> samples[ :int(len(samples)/2)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> samples[ :int(len(samples)*.25)]
[0, 1, 2, 3, 4]
>>>
>>> samples[int(len(samples)/2): ]
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
>>>
>>> samples[int(len(samples)*.75): ]
[15, 16, 17, 18, 19, 20]
```

Reversing the Samples

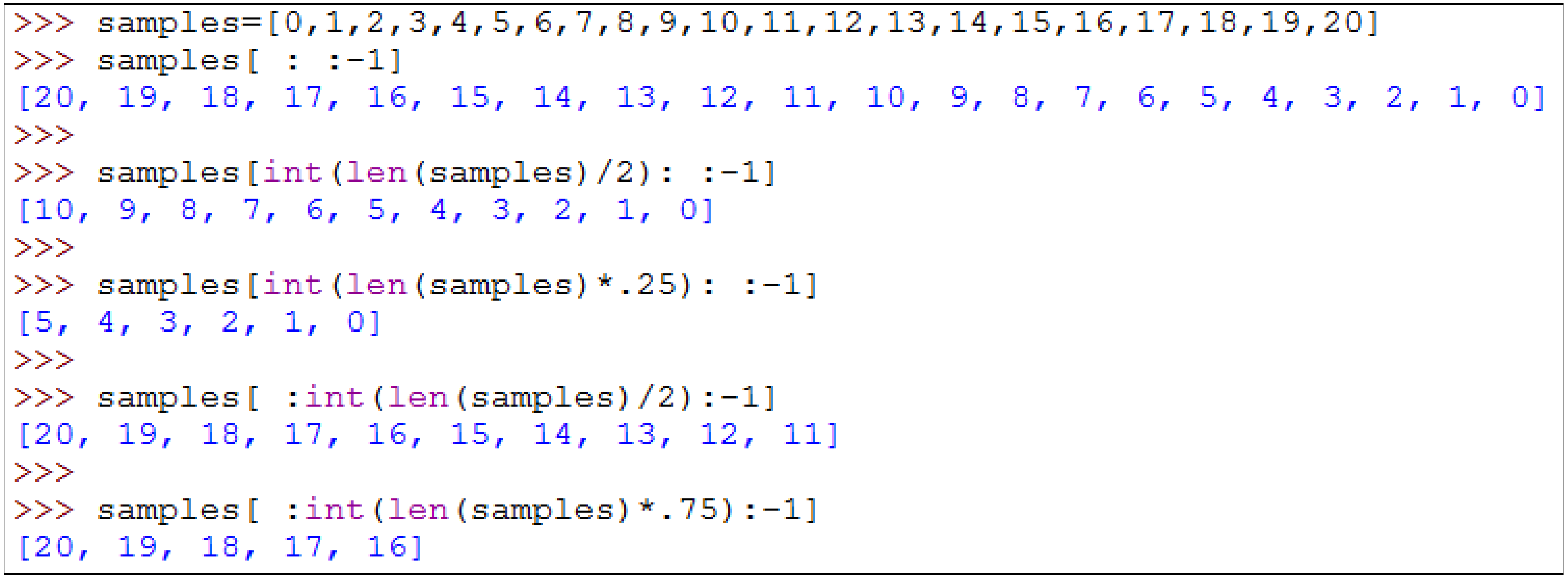
```
# Reverse all the audio
samples[ : :-1]
# Reverse the first half of the audio
samples[int(len(samples)/2): :-1]
# Reverse the first 25% of the audio
samples[int(len(samples)*.25): :-1]

# Reverse the last half of the audio
samples[ :int(len(samples)/2):-1]
# Reverse the last 25% of the audio
samples[ : int(len(samples)*.75):-1]
```

No end number is given, so Python will stop at the beginning, because the '-1' shows you want to go backwards

No start number is given, so Python will start at the end, because the '-1' shows you want to go backwards

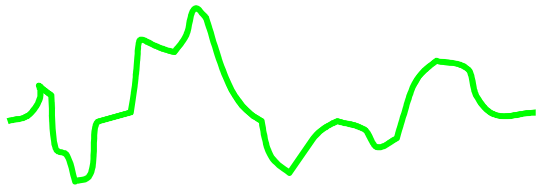
Trying it in the Shell



```
>>> samples=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
>>> samples[ : :-1]
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> 
>>> samples[int(len(samples)/2): :-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> 
>>> samples[int(len(samples)*.25): :-1]
[5, 4, 3, 2, 1, 0]
>>> 
>>> samples[ :int(len(samples)/2):-1]
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11]
>>> 
>>> samples[ :int(len(samples)*.75):-1]
[20, 19, 18, 17, 16]
```

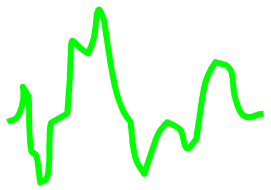
Playing the Audio at Faster Speeds

The original audio



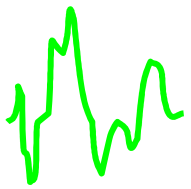
```
# Access every second sample of the audio.
# If you listen to the result, it
# will be twice as fast.
samples[ : :2]
```

Keeping every second sample



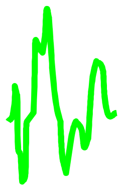
```
# Access every third sample of the audio.
# It will be even faster than the previous example.
samples[ : :3]
```

Keeping every third sample

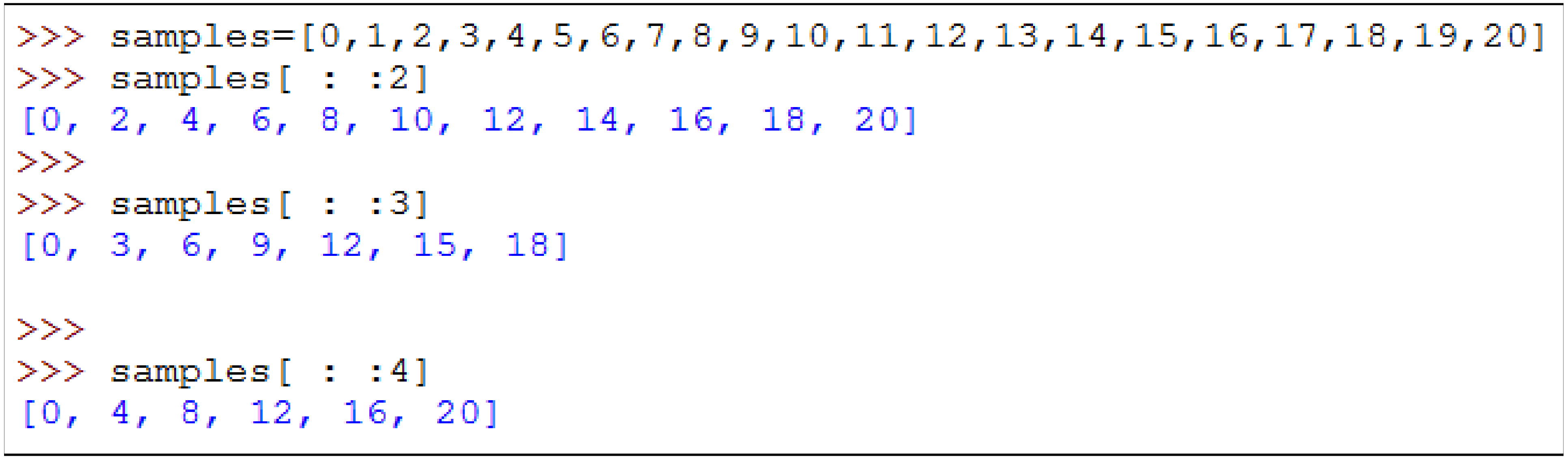


```
# Access every fourth sample of the audio.
# It will be even faster than the previous example.
samples[ : :4]
```

Keeping every fourth sample



Trying it in the Shell



```
>>> samples=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
>>> samples[ : :2]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
>>>
>>> samples[ : :3]
[0, 3, 6, 9, 12, 15, 18]

>>>
>>> samples[ : :4]
[0, 4, 8, 12, 16, 20]
```

Rounding a Float to an Integer Number

- If you want Python to round a float up/down to the closest integer, one way is to use `round()`
- However, if the float is in the middle of two integers e.g. 1.5 Python will *round to the nearest even integer*
- We haven't used `round()` in any of the examples discussed in this presentation, but you might find it useful later

```
for i in range(0, 11):
    print(i+0.5, "becomes", round(i+0.5) )
```

```
0.5 becomes 0
1.5 becomes 2
2.5 becomes 2
3.5 becomes 4
4.5 becomes 4
5.5 becomes 6
6.5 becomes 6
7.5 becomes 8
8.5 becomes 8
9.5 becomes 10
10.5 becomes 10
```