MODULE 02 - 059: Handling `slice()` to Store Slices in Python

Introduction to `slice()` in Python

In Python, slicing allows you to extract portions of a list using indices. The `slice()` function provides an alternative way to achieve this dynamically, making it particularly useful when dealing with reusable or programmatically defined slices.

Why Use `slice()` Instead of Explicit Slicing?

Reusability – Store slice parameters in variables for repeated use.
Dynamic Slicing – Adjust slice parameters programmatically.
Improved Readability – Reduces clutter when working with complex slices.
Enhanced Debugging – Easily inspect slice properties (`start`, `stop`, `step`).

Python Documentation: slice() function

Basic Usage of `slice()`

```python
# Sample List
languages = ['Python', 'JavaScript', 'Ruby', 'C++', 'Java']

# Using explicit slicing
print(languages[:2])  # Output: ['Python', 'JavaScript']

# Using slice() function
slice_obj = slice(2)
print(languages[slice_obj])  # Output: ['Python', 'JavaScript']
```

Explicit slicing (`[:2]`) and **slice object (`slice(2)`)** produce the same result.
The `slice()` function returns a **slice object**, which can be reused.

Python Documentation: List Slicing

Understanding `slice(start, stop, step)` Syntax

The `slice()` function accepts up to three arguments:

Argument	Description
`start`	Index where the slice begins (inclusive). Defaults to `None` (start of list).
`stop`	Index where the slice ends (exclusive). Defaults to `None` (end of list).
`step`	Number of elements to skip (default is `1`).

```python
# Sample List
numbers = [10, 20, 30, 40, 50, 60, 70]

# Explicit slicing
print(numbers[1:5:2])  # Output: [20, 40]

# Equivalent using slice() function
slice_obj = slice(1, 5, 2)
print(numbers[slice_obj])  # Output: [20, 40]
```

Python Documentation: More on Lists

Using `slice()` for Dynamic Slicing

The `slice()` function is especially useful when working with **dynamic indices**. Instead of hardcoding values, you can generate slice parameters programmatically.

```python
# Function to get the first N elements of a list

def get_first_n_elements(lst, n):
    slice_obj = slice(n)
    return lst[slice_obj]

numbers = [100, 200, 300, 400, 500]
print(get_first_n_elements(numbers, 3))  # Output: [100, 200, 300]
```

Instead of manually typing `[:3]`, we pass `n` dynamically.
This approach enhances **flexibility** and **code maintainability**.

Python Documentation: Built-in Functions

Retrieving `slice()` Object Properties

You can inspect a `slice` object's properties using `.start`, `.stop`, and `.step` attributes.

```python
# Creating a slice object
slice_obj = slice(1, 6, 2)

print(slice_obj.start)  # Output: 1
print(slice_obj.stop)   # Output: 6
print(slice_obj.step)   # Output: 2
```

Useful for debugging and analyzing slice behavior in **complex algorithms**.

Python Documentation: slice Attributes

Best Practices When Using `slice()`

Prefer Explicit Slicing for Simple Cases – Use `list[start:stop:step]` for quick operations.
Use `slice()` for Reusability – Store and reuse slices when working with large datasets.
Leverage `slice()` in Functions – Allows for **dynamic slicing** based on function parameters.
Combine `slice()` with Iteration – Use it effectively in loops and comprehensions.

Python Documentation: Slicing Best Practices

Summary & Key Takeaways

`slice()` provides an alternative to explicit slicing (`[:x]`).
Supports three arguments: `start`, `stop`, `step`.
Enhances code reusability and flexibility in data processing.
Useful for debugging: Access slice properties via `.start`, `.stop`, `.step`.
Ideal for handling dynamic slicing in large-scale applications.

Python Documentation: Complete Guide to Slicing

Video lesson Speech

So far in this section on lists in Python we've covered a number of different ways to slice and to work with different ranges inside of python and you may think that I'm going over the top with it because we've covered it from so many different angles and it really does boil down to this type of process.

large

Figure 1: large

large

Figure 2: large

The process of slicing and working with lists is absolutely critical and you're going to be doing it on a daily basis when it comes to implementing machine learning algorithms and even building and working with data in web and mobile applications.

So, I want to make sure that we have it covered and working with slices becomes second nature to you.

With that being said I'm going to now show you another approach to doing that.

So far in this course, we've worked with implementing slices like this where I could say print and tags and grab the first few elements all the way to ending with the second index which will give us python and development

And this works fine.

But there are times where you may not know or you may not want to hard code in this slice range.

And so in cases like that Python actually has a special class called slice which we can call and store whatever these ranges we want are and what they're going to be.

So I'm going to comment this out and let's talk about the slice class.

So with what we have here, we can create an object.

I'm going to call it a slice object and store it in a variable but you can name it whatever you want and the syntax for creating a slice object is simply providing the keywords slice and then passing in a number of arguments.

And this is going to be one of the main topics we talk about in this guide which is the slice class has all kinds of different variations on how you can call it and what you can pass to it.

I want to show you one of the most basic ways to do it and it's going to give us the same result that we have from our first example.

So if I call slice and just pass in two and just try to print out the slice object you're going to see it doesn't actually return the same result set it returns an object.

But then we have access to that result set through the object and I know that probably makes no sense whatsoever so let's walk through the example so you can see how it works.

If I run this you can see that all we get back is what looks like little function call where it says **slice(None, two, None)** :**

And **this gives us a little bit of a hint on what is available to us**.

But let's take a look at what we can do first in order to retrieve the results set.

So, the way we can do it is very similar to what we did here.

I can just call tags and then pass in using the same bracket syntax pass in our slice object.

Now if I run this you can see we get python and development.

So now that you see it working.

Let's talk about what exactly is going on behind the scenes and how you can extend this further.

large

Figure 3: large

large

Figure 4: large

large

Figure 5: large

So, what we have right here is very similar to what we have here when we were working with an explicit slice and when we passed in a range the only difference is that, I should say, **the key differences here is we can call store this method inside of another object inside of a variable and then we can call that anywhere in the program**.

Instead of doing something like what we have right here if I tried to do this and pass it and you'll see that I get an error.

And so t**his is a very nice way of being able to store your slice so that you can reuse it on any other kinds of lists**.

So, if I run it again, you can see everything is back and working.

Now, that we have that let's talk about the different variations and how you can call this, I'm going to print out that object again.

And let's see what we have access to.

---

slice(start : stop : step) syntax

So with **slice**, you can see there are **three potential arguments inside of this object**.

1. The first one is our start point. So we're going to have a start.
2. We're going to have an end.
3. Then we're going to have a step which if you remember exactly with what we had with ranges.

With these explicit type of slices we could pass in say [2:4:2]:

And then this is going to bring us every other element because the last 2 is our step. This first 2 is our start and this 4 is our stop or this is our endpoint.

This is exactly what we have access to with the slice class.

So, what we could potentially do with slice is say that we want a start at the second element. I'm going to say one. And I want to go to let's say the Fifth Element and actually there is not a fifth one in there. There's only four.

Because remember these are index values so 0 1 2 3 4.

And then I want the step to be 2. And just so you believe me.

Let me replicate this here some say 1 4 and 2.

And let's make sure that these are matchings.

I'm going to get rid of our slice object.

We're going to have the first set.

Print it out using our explicit type of syntax and the second one using a slice object. And if I run this you can see we get the exact same result set.

We get development because we started at one then it skips because we said there's going to be a step here that is two.

And so it grabbed this code and then it ended right at the four.

And so **that is doing exactly the same thing**.

large

Figure 6: large

large

Figure 7: large

large

Figure 8: large

It's simply a functional approach as opposed to an explicit type of approach and working with this type of object also gives us a few other little helper functions.

Use case Say that you're working on a machine learning algorithm and you want to know in some other part of the program where the range started where it stopped and what the step interval was.

And you may think this is kind of pointless because in this example we could simply look on line 11 and we can see all those values and that's 100 percent true.

But I want you to think in a little bit more of a production application there are going to be times where an algorithm might return a slice and you have no idea what the start, stop and step points are and so when you can work with this type of function it's really nice because I can say slice object and then call start and what this is going to do is it's going to tell me the first index so this is going to tell me it started at 1

Then, if I want to print out when it's ending it's going to give me the last index which is four.

And then, if I want to see what the step is I can do step run that and you can see this gives us 1 4 and 2.

which is exactly what got placed here.

Once again, there's not much of a point to it.

When you're the one calling slice, however, there will be times where algorithms return slice objects and this can be very helpful to understand exactly what it's doing.

One of the most helpful ways that I've been able to use it is by being able to call step on a slice object because then that can tell me right away what types of intervals that the program is returning the data with and so that is something that I have practically been able to use myself.

So in summary what we've walked through is an alternative way to define and call slices.

One of the biggest reasons why you'd ever use this slice class over using just this explicit version slice(1, 4, 2) is whenever you want to define your ranges and your steps and those kinds of elements and you want to store them in a variable and then simply call them later on and or also another opportunity where this would be a very good fit is if you're maybe calling this on different datasets.

So, say that you have `tags.`, another one might be something like `site_wide_tags` or something like that and you want to implement the same process you want to grab the same slice elements on different types of data structures and with different data.

But by being able to save your slice you don't have to duplicate your code.

And then if you ever want to make any other changes in the future you only have to make them one time instead of having to make them in every single spot where you defined this type of range.

You're going to be using both of these options quite a bit especially if you're going into the machine learning and data science side of the world.

But it is good to have a good understanding of when it's the best opportunity to use one versus the other.

---

large

Figure 9: large

large

Figure 10: large

Code

```
# 02-059: slice() - Handling slice() to store slices.

tags = [
    'python',
    'development',
    'tutorials',
    'code',
    'programming'
]
print('\nOriginal List: ' + str(tags))

print(tags[ : 2 ])      # ['python', 'development']


# slice() - How Slice works:
# slice() returns a tuple, adding one None items on the right/on the left both sides.
## Explicit slice()

slice_obj = slice(2)
print(slice_obj) # slice(None, 2, None)

## Explicit slice() using a list

slice_obj = slice(tags)
print(slice_obj) # slice(None, ['python', 'development', 'tutorials', 'code', 'programming'], None)

## Slice() as an index onto a new var

slice_obj = slice(2)
print(tags[slice_obj])  # Again, ['python', 'development'], using slice(2) as a new object on index


## slice() and Immutability
## (Uncomment this to see what syntax error is returned)

# slice_obj = [ : 2]
# print(tags[slice_obj])  # Slice_obj, which stored "slice(tags)" cannot be reused like this


# slice( start : stop : step ) syntax

print(tags[1 : 4 : 2])      # ['development', 'code']

slice_obj = slice(1, 4, 2)
print(tags[slice_obj])      # Both methods returns the same, ['development', 'code']

print(slice_obj.start)      # Slice Start:  1
print(slice_obj.stop)       # Slice Stop:   4
print(slice_obj.step)       # Slice Step:   2

# More examples

list2 = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
```

```
print('\nOriginal list:  ' + str(list2))

print(list2[ 2 : -1 : 3 ])  # c, f, i

slice_list2 = slice(2, -1, 3)
print(list2[slice_list2])   # c, f, i

print(slice_list2.start)      # Slice Start:  2
print(slice_list2.stop)       # Slice Stop:   -1
print(slice_list2.step)       # Slice Step:   3
```