

FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture40

Lecture 40: Generics in Python

Welcome to lecture 40. Templates and generics: so far, we have seen template functions, template classes in C++, and generics in Java. In the last class, I introduced that in Python, we also have generics. So, when we look at the simple code, right? In fact, in the last lecture, I had executed a few programs if you recall. All right, so continuing on... So, suppose you have a conditional statement. You know, in C++ and Java, you know very well it is similar, right? It is similar to Java or C++. So, slightly, you can see some syntax changes, right. So, here you go. Suppose you want to use the conditional statement, right? Assume that x is equal to 5.

If x is greater than 0, you are going to print 'it is positive.' We use 'else if.' So, here, 'else if' is 'elif,' right. x equal to 0, else if equivalent: x double equal to 0, that means if it is equal to 0, print '0,' otherwise, else print 'negative,' OK. So, the syntax is very similar to C++ or Java, as you have seen, right. So, if you practice Python, it will be nothing for the initial part of the code, and then when you go deep into Python...

I mean, since you know the complete syntax of Java and C++, it will not be very difficult. Okay. So, this is one case: a conditional statement. So, if I execute this code, right, since x is equal to 5, right, if x is greater than 0, which is true, therefore, 'positive' is getting printed. Another thing: you want to use a loop, right.

In the case of loop, for example, here, for i in range of 5, print i. So, the index here is starting from 0. It is equivalent to for i is equal to 0 i less than 5 i plus plus right and then c out i right. I will write the equivalent code because you are all familiar with c plus plus for i is equal to int and assume that I have declared int i is equal to 0 i less than 5 i plus plus right c out i. So, you know what you will get

we will get 0 1 2 3 4. Similarly, here So, when I write this in python for i in range of 5 print i all right.

Control Structures in Python

- Python uses indentation to define code blocks.
- Control structures include if, for, and while.

Example of Conditional Statements:


```
1 x = 5
2 if x > 0:
3     print("Positive")
4 elif x == 0:
5     print("Zero")
6 else:
7     print("Negative")
```

Output
Positive

*for (int i=0; i<5; i++)
 cout<<i;*

Example of Loop:

```
1 for i in range(5):
2     print(i)
```



So, now, we since you know this. So, we will be getting 0 1 2 3 right. So, this is the basic structure you can practice I mean till loop I mean it would not be very difficult. So, just take the interpreter whatever interpreter you like in fact, last lecture I introduced And today, in fact, we can see with the help of command prompts.

So, like Java and C++, so this is also an object-oriented programming, right? Python is also OOP. So, we have to know the basic syntax of the classes and objects, right? So, for example, in this code, you are seeing class person followed by colon. Whereas, in Java and C++, we see followed by the begin, right?

The begin operator, like the curly braces. Right. So here we go with a colon. And then here you have double underscore init. Right.

And then again, double underscore. So this is a special member function. Exactly like a constructor. Right. So that we have seen in either Java or C++.

And then you have the instance variable self, like this pointer in C++ or Java. All right. And then you have, let us say, two arguments you are passing. So, I have two parameters, and then, you know, we have studied this dot name equal to name in Java several times. So, here self dot name equal to name, and self dot age is equal to age, and then you have a member function, right.

So, here you have a member function you are defining greet all right. So, this self pointer is going you know that this I can pass always this and then here you are having because self I am passing because I have to use self dot name. right hello my name is self dot name just recall what we have studied in java i mean maybe here and there we have to change the syntax otherwise both look almost same right and then you are having a object let us say person is object right and here you have the parameters pausing allies and 30 all right you are pausing allies and 30 so once you are pausing right so what will happen so this is like as i said this is like a constructor all right in fact it is a special member function So, name is allies. Right.

So, name will be assigned to self dot name. Age will be assigned to self dot age. Right. So, that will be returned to your P. right.

So, that will be returned that is what you are writing right. So, that will be returned to your p and the next line object p right which is returning returning means this is object right. So, in fact self dot name is allies and self dot age is 30 right. So, this object p is invoking greet. So, when it is invoking greet it will print hello my name is you know self dot name is allies all right.

So, I am not printing any age. So, when I run the code maybe this code what we can do we can run it in command prompt. So, just we will go to command prompt. So, in the command prompt if I run this particular code. So, this is a command python 3 the name of the file yeah.

So, person underscore class dot py. So, this file has been stored as person underscore class dot py. So, when I run the code I am getting hello my name is alice. So, this is what we have expected in fact when we go to the slide. So this will be your output, right?

which is relying on the typing module and then we are going to have the explicit type declaration. So, here suppose I want to right. So, that is typing import list and type variable and type variable we are going to use t. So, this t for example, I am creating an array right.

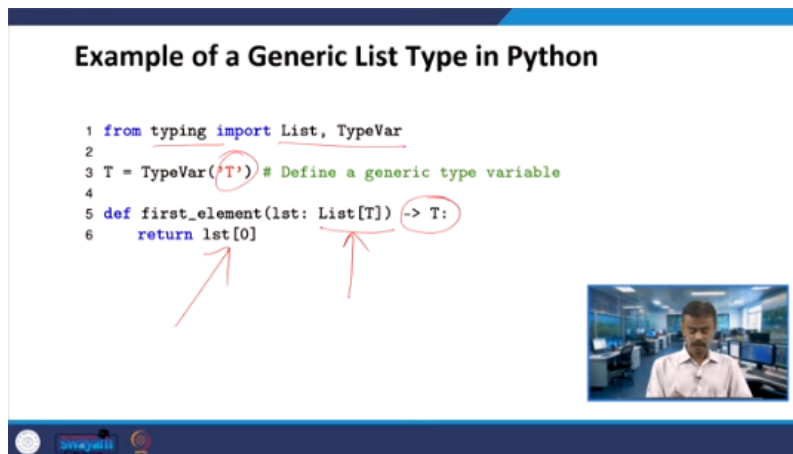
So, I have an array all right. So, here in Python the return type t this is called the return type t and it is returning the first element this is a simple example. Right. And it is returning the first element. So just recall how do you write the same code in C++.

You put t first and first underscore element and then you are passing an array. Right. You put t. List of t. Right. So or list of t whatever be the return type. Then you are writing return.

So here I mean more or less I mean you can understand if you practice it is not very difficult. So the list of t that array will be that base address. Right. That is what I am talking in terms of C++. right the equivalent so if i write like this so the base address lst right and then if i put lst of 0 so that means it will have lst 0 lst 1 lst 2 etc are in the case of c++ if i simply print lst it will print all the all the elements right so here if i put lst of 0 the first element will be printed so that you are returning that is what exactly this program is doing

Example of a Generic List Type in Python

```
1 from typing import List, TypeVar
2
3 T = TypeVar('T') # Define a generic type variable
4
5 def first_element(lst: List[T]) -> T:
6     return lst[0]
```



So, the main important point is if you look at the typing, all right. So, the typing module and you are explicitly you are going to use or you call it as a generic type variable. The t is a generic type variable, right. We will see an example how we can use this. So, for example, up to this I have told whatever we have done in the


previous slide and the usage point of view assume that I have a integer list int underscore list.

equal to 1, 2, 3, right? And string underscore list, apple, banana, cherry. So, now print, right? So, what you are doing? Print, you are calling get first element, right?

Example: Type Hints and Generics in Python

A generic function to return the first element of any list type.

```
1 from typing import List, TypeVar
2
3 T = TypeVar('T') # Define a type variable
4
5 # Generic function to get the first element of any list type
6 def get_first_element(list: List[T]) -> T:
7     return list[0]
8
9 # Usage
10 int_list = [1, 2, 3]
11 str_list = ["apple", "banana", "cherry"]
12
13 print(get_first_element(int_list)) # Output: 1
14 print(get_first_element(str_list)) # Output: "apple"
```



So, this is what we define. Here, I have defined first element, first underscore element. So, here, It is a generic function. This is an example of a generic function, right?

Get underscore first element. Why I am calling generic function? I am using t and return type is also t, right? So, when I am calling this function, get underscore first underscore element and I am passing int underscore list, right? So, assume that I am passing the base address of this, correct?

So, when it goes here, you are passing 1, 2, 3. Now, T is nothing but an integer list. Capital T is an integer list. Correct?

So, it has 1, 2, 3. And then it is going to integer. Once it becomes T is integer, this is also integer. It is going to return an integer. And what are you returning?

Return list 0. That means integer list 0. Index 0. Index 0 is what? 1.

Correct? So 1 will be the output. The first case, I will write here. We will see. And similarly, next one it is becoming string list.

Character array. Right. This is 0, string 1, string 2. Right. So, in fact, string array.

I said character array, string array. String 0, string 1 and string 2. Right. So, you are passing this and get underscore first underscore element. Now, the list will be nothing but string underscore list.

Exactly like what we have done with templates in C++ or generics in Java. Right. Now, this T. Capital T is nothing but a string underscore list. Right.

So that means you have passed apple, banana, and cherry. Right. So the return type will also be a string list. You are trying to print. I mean, you are trying to return a list of zero.

That means. The zeroth index. What is the zeroth index? Apple. All right.

So, the next output I have to get is apple. All right. So, this is the output I will get. So, when I run the code. All right.

So, this is the output. Maybe we can also try this in the command prompt. And when I go to the command prompt. All right. So, this is the code I will be running.

So, you can see python3 get_underscore_first_element.py. So, get_underscore_first_element.py is your Python code. All right. So, whatever I have written here. The one you are seeing on the slide will be saved as this.

And then, when you are running the code, you can see you are getting the output 1 and apple. Right. So, that is what we are also expecting. And in fact, when I run the code, I got this 1 and apple. Right.

So, one is the integer list. Another one is the string list. Right. So, this is for your reference. So, how this is working.

So, get_underscore_first_element is a generic function. And I have used t as a type variable. So, this type variable one time it is working as an integer, another time it is working as a string, right? The code reusability. So, I do not need to write that, which is what the beginning of this particular chapter I explained. So, we will see some of the other cases.

So, now we will see the container class. So, a container class is nothing but a generic. We call it a generic class in Python. If you look at the title, this is an example. So, the generic container class is useful to store and retrieve any data type.

So, for example, we have already seen from typing, you are importing type variable and generic. All right. And then, we have already defined a class container. All right. Generic is a keyword, and T is a type that you are going to use.


And also, we have seen the special member function double underscore init double underscore. Like star this, we have a self pointer over here. All right. Which is, in fact, an instance of a class and a value. Right.

So T integer means it will be integer. If it is a string means it is a string. So, then you have self dot value equal to value this we had already seen the age and name self dot name equal to name. So, like this you are writing here and you are having one member function get value. So, self all right and then return type is t. So, which is returning self dot value all right.

Example: Generic Class in Python

A generic container class to store and retrieve any data type.

```
1 from typing import TypeVar, Generic
2
3 T = TypeVar('T') # Define a generic type variable
4
5 class Container(Generic[T]):
6     def __init__(self, value: T):
7         self.value = value
8
9     def get_value(self) -> T:
10        return self.value
11
12 # Usage
13 int_container = Container(123)
14 str_container = Container("Generics in Python")
15
16 print(int_container.get_value()) # Output: 123
17 print(str_container.get_value()) # Output: "Generics in Python"
```



So, now you have the container. right. So, container and you are passing 1, 2, 3. 1, 2, 3 is what? Integer, correct.

So, you have integer container, object is integer container, right. So, when you are passing 123, your generic of t, correct. So, that will be nothing but 123 because it is a integer, correct. So, here you go when you are having your special member function in it, So, whatever value you are passing 123 will be copied into self dot value, all right.

So, which is nothing but self dot value. And similarly, you have a string underscore container. So, again you are calling the container, but this time you are passing generics in Python, a string you are passing. So, now this capital T

will be string now, string underscore container. And value is nothing but generics in Python.

Right. So that will be stored as a self dot value. So now when you are printing when in container this object is calling the function get value get underscore value. So here you go self dot value initially for the case of integer underscore container it is 123 will be the output. Right.

And the second case. String underscore container object when it is calling or invoking the function get underscore value you should get generics in Python as a output this one will be the output. So when I run the code all right so you will get the output like this. So this you try right the two programs we have done with the help of command prompt so what you have to do is a practice. right.

So, just find out right or just see how I have run the code, and then this I am giving as an exercise, kind of right. So, try it in the command prompt, all right. So, in the previous program, in the previous slide. So, the container, which is nothing but a generic class, and also I told. So, this can store a value of any data type, all right, and then, yeah, as usual, when I am using T.

So, this is a type variable, and this allows containers to be more flexible. So, that means it can use different data types. So, generic classes can improve type safety, right, because you are using various types. For example, in the last few cases, we have seen integer type and string type, like C++ and Java, right. So, you have type safety and reusability.

Right. So, you may have, like in the previous example, one you are using for integer, another one you want for string, right. So, that can be handled exactly like what we have seen in Java. So, maybe we will have one more example, all right. So, this is a function to merge two dictionaries with generic keys and values, right.

So, as usual we have here, here we are going to have a dictionary and type variable and k we are using V we are using type variable we are defining K and V it is like a template class you define S comma U if you recall in C plus plus right. So, now I define a function merge underscore dictionaries all right. So, you use a dictionary right you call it as a dictionary 1 and use another dictionary call it as a


dictionary and return type if you look dictionary K comma V right. It is like adding two complex numbers.

C1 is one complex number, C2 another complex number. Addition will give you the third complex number. So, here the return type is another dictionary in the form k comma v. So, here this dictionary 1, right. So, let us say it is invoking the copy function, right. So, assume that it is invoking the copy function.

Example: Generic Function with Dictionaries in Python

A function to merge two dictionaries with generic keys and values.

```
1 from typing import Dict, TypeVar
2
3 K = TypeVar('K') # Key type
4 V = TypeVar('V') # Value type
5
6 def merge_dictionaries(dict1: Dict[K, V], dict2: Dict[K, V]) -> Dict[K,
    V]:
7     result = dict1.copy() # Start with the first dictionary
8     result.update(dict2) # Merge in the second dictionary
9     return result
10
11 # Usage
12 dict_a = {1: "apple", 2: "banana"}
13 dict_b = {3: "cherry", 4: "date"}
14
15 merged_dict = merge_dictionaries(dict_a, dict_b)
16 print(merged_dict) # Output: {1: 'apple', 2: 'banana', 3: 'cherry', 4: '
    date'}
```



So, that means it will start with the first dictionary. It will start with the first dictionary. And the result, right? So, you have a result. Now, result is getting updated with dictionary 2.

So, that means the first dictionary will be merged with the second dictionary, right? So, that is what this program is going to do and then this function will do and it is going to return result, right? So, first dictionary you are copying, second you are updating, right? So, we can see an example. Here you have dictionary A, you have 1 is apple, right.

So, dictionary 1, you have apple and 2 is banana, right. Dictionary B, you have 3 cherry, 4 date, right. So, now I am calling this merge dictionaries under the object merge. It is going to return to merge underscore DICT, right. You are passing dictionary A and dictionary B. That means 1 apple, 2 banana, we are passing.

Similarly, 3 cherry, 4 date will be passed, right. So, now when you have This print merge dictionary. All right. Merge underscore DICT.

Yeah. Because here it is returning. When you are calling this merge underscore, you are calling here. And then you are passing dictionary underscore A, dictionary underscore B. So, that means it has to return result. Correct.

So, this result will be stored in merge underscore DICT. And when you are printing merge underscore DICT. Right. So, both will be merged all right your merge dictionary will be now one apple two banana three cherry and four date right so that you will get an output all right maybe we can run this code okay so that in the previous program you will get the confidence so here what i am doing i will just run this code in the command prompt we will go to the command prompt and then python 3

the name of this program is merge underscore dictionaries dot python py and you can see the output right. So, the same code whatever you have used here the same code if I run you are getting whatever the output we were expecting this output we get right. So, here merge underscore dictionaries right so we define that is a generic function so that will work with any type all right so in fact with any key and type value so here we have more than one type variables k and v previously we use only t right so here one is the key another one is value so k and v that we are using one is the key another one is the value in fact the common from title key value value key type value type right.

So, we use k and v here. So, this shows right. So, this particular function shows Python's flexibility in handling mapping with type safety in a similar line. So, we can see this particular program. So, here we are going to use the tuple. Right? What do you mean by tuples in Python?

So, I use from typing, all right. So, this will import tuple and type variable. So, I am going to use type T. So, here I define the generic function swap_tuple, all right. So, here you have tuple T comma T, all right.

So, which I am mapping with par, meaning the base address is par, and the return type is tuple T T, all right. So, what is it returning? So, par, when it is returning tuple T T, returns to par. You will have par 0, par 1, etcetera, right. So, whatever the numbers that we are taking, in this case, it is a tuple. So, you have 10 comma 20.

For example, here, in fact, I have taken the same example, 10 comma 20, right? So, what is meant by this? When I am assigning over here, it will get pair of 0 as 10, pair of 1 as 20, right? So, that is the meaning, okay? So, what we are doing here is returning pair 1 first and then pair 0.

So, for example, I define integer pair 10 comma 20. So, when I do this, it will be assigned to pair. That means pair of 0 is 10, pair of 1 is 20. So, now I am returning. I am returning in the form of another tuple.

It is like a complex number 1 plus complex number 2. C3 will be another complex number. So, I am returning a tuple. So, when I am returning a tuple, I want to return pair 1 first and then pair 0. That means swapping.

So, what is the output you will initially get? 2010. Alright. So, that is when you call this. Right.

When I call this line number 12. Print swap underscore tuple. That is calling the function. You are passing int underscore pair. That means int underscore pair 0 and int underscore pair 1.

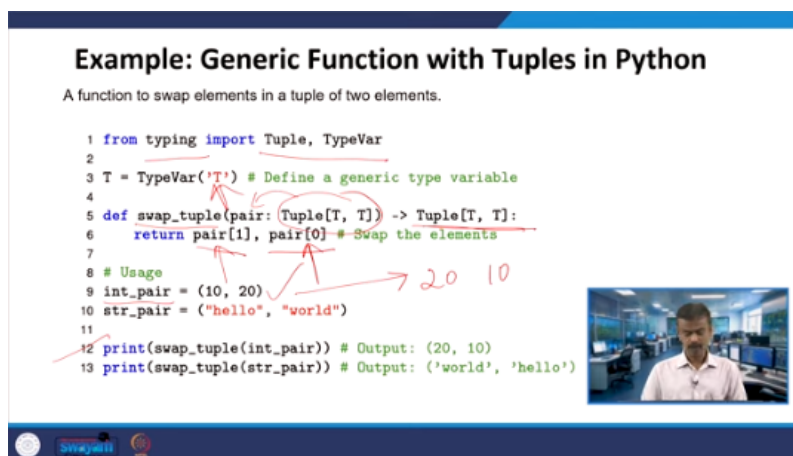
That will be mapped to. That means the base address pair. Pair of 0. Pair of 1. Already said.

Pair of 0 is 10 and pair of 1 is 20. That is getting swapped. So, you will get initially the output this. Second one, you have string pair, hello world, right. So, when it goes here, now T is string, tuple.

Example: Generic Function with Tuples in Python

A function to swap elements in a tuple of two elements.

```
1 from typing import Tuple, TypeVar
2
3 T = TypeVar('T') # Define a generic type variable
4
5 def swap_tuple(pair: Tuple[T, T]) -> Tuple[T, T]:
6     return pair[1], pair[0] # Swap the elements
7
8 # Usage
9 int_pair = (10, 20)
10 str_pair = ("hello", "world")
11
12 print(swap_tuple(int_pair)) # Output: (20, 10)
13 print(swap_tuple(str_pair)) # Output: ('world', 'hello')
```




Example: Generic Function with Tuples in Python

A function to swap elements in a tuple of two elements.

```

1 from typing import Tuple, TypeVar
2
3 T = TypeVar('T') # Define a generic type variable
4
5 def swap_tuple(pair: Tuple[T, T]) -> Tuple[T, T]:
6     return pair[1], pair[0] # Swap the elements
7
8 # Usage
9 int_pair = (10, 20)
10 str_pair = ("hello", "world")
11
12 print(swap_tuple(int_pair)) # Output: (20, 10)
13 print(swap_tuple(str_pair)) # Output: ('world', 'hello')

```



You are passing hello, one tuple is hello, another one is world, right. One component is hello, another one is world in the tuple. So, pair of 0 is what? Hello. Pair of 1 is world, ok.

So, when you are returning this, the other way around you are returning, pair 1 and pair 0. that is a tuple all right. So, when you are calling the function and you are passing this string path you have to get the output the other way around world and hello all right world first then hello ok. So, when I run the code this you can try right when you run the code you have to get the output like this ok. So, this is a generic functions

With the tuples in Python, yeah, the same explanation. This is for your reference. Swap underscore tuple is a generic function, right? And then, when you are using this generic function, so this has the flexibility of using integer, string, or any other data type. In this case, we had seen integers and strings, all right? And the type safety is obviously maintained through. Python's type hinting, and we have one more function, right? Let us compute the Euclidean distance. What do you mean by Euclidean distance in Python, right? So here, you are importing math. The rest you know, from typing, you are importing list and type variable. So here you go, right? So I am going to define a type variable for integer or float, right? So, type where T, I put integer, comma, float, right? And then I define a generic function Euclidean underscore distance. So, 0.1 list T, you are passing. You are mapping to point 1 and another list you are passing, map to point 2, all right.

So, you have x1, y1, x2, y2, and I am asking you to find out the Euclidean distance, right. $x_2^2 - x_1^2$ the whole square plus $y_2^2 - y_1^2$ the whole square, and then you are taking a square root, right. So, here the Euclidean distance, it is

like the length of the point, right. So, suppose I take x_1, y_1 , right? Or x_1, y_1, z_1 , three dimensions. Here also, it should be x_2, y_2, z_2 .

Right? Otherwise, suppose your input is x_1, y_1, z_1 , another one is, let us say, x_2, y_2 only, all right? Assume that one is a 3D point, another one is a 2D point. You cannot find any distance, right? That is an error, so you raise the error, right? So what if the length is not equal? I gave an example: one is a three-dimension, another one is two-dimension. So then, points must have the same dimension. That means both should be the same dimension. If so, right, if length of point equal to length of point 2, then return the square root of this, right. So, as I said, x_2 minus x_1 the whole square plus y_2 minus y_1 the whole square. So, if length of 0.1 is equal to length of 0.2, right.

So, in that case, so we know that if both the dimensions are equal, suppose I have, I will take only two dimensions. So, we know, right, x_2 minus x_1 the whole square plus y_2 minus y_1 the whole square and then the square root, right. So, square root of math square root of math dot square root of. So, this is a summation, right, plus you have. Suppose you have, let us say, more dimensions.

right. So, summation x minus y . So, this is your x and this is minus y , ok, x minus y , the whole square, right. So, that is what the notation here. And then you are zipping into 0.1 comma 0.2, putting into 0.1 comma 0.2. So, now we will take an example.

Suppose I have 1, 2, 3, point A is 1, 2, 3 integer, point B is 4, 5, 6. So, which is also an integer. And then I am calling the function Euclidean underscore distance, the generic function, passing point A and point B. Passing two lists two arrays point A and point B. So, when I pass this it is calculating this Euclidean distance that is what we have written all right. So, one array will go as X another array will go as a Y all right.

So, that is what happening over here when it is finding out the Euclidean distance. All right. And then you are going to get the output. So, first output we are expecting 5.196. And in the second case you can see it is a float.

Right. Point C and point D float. So, I am passing this now. Euclidean underscore distance. I am passing point C and point D. Right.

The same formulation. This time it is a float. And we will get the output like this. Okay. So, you can check this.

How it is working? You can check this. In fact, 0.5 only we incremented in all the cases right. So, both should be same right when you are subtracting let us say 4.5 minus 3, 5.5 minus 2.5 3, 6.5 minus 3.5 3 ok.

So, both you have to get the same output. So, this also you can try. So, in fact, we can get the output like this when you run the code right. So, this is the complete explanation how the generic functions are like the previous example.

So, this is for your reference you can just go through that. So, whatever I have explained in the previous slides. So, each and every modules. So, I have given as a explanation over here. So, the conclusion point of view.

So, when we are using C++ we use template right when we use C++ it was template right. So, in the case of Java and Python we call it as generics, all right. So, in the case of C++, we have already seen, it is a compile time generic programming and it is highly flexible, but it is lacking the strict type safety, all right. So, when you are using integer or double or float, there is some, right, I mean, in fact, it is lacking strict type safety.

So, the templates can lead to larger binaries. So, this we had seen, in fact, when we are talking about So, converting and then the reconstruction if you recall right. So, the binaries right. So, because of due to separate instantiation for every data type right.

So, this is about C++ template. So, whereas, in the case of Java you have the strict type safety. All right. So, the compilation time it is checking the type safety, but it is limited by type arrays. All right.

So, in fact, when we talked about Java generics, we had seen these properties, and it also requires bounded types. Correct. And wildcards to support complex generic types. So, these are all we had already seen in the case of C++ and templates. And in the last few lectures.

All right. So, we talked about Python generics, all right. So, this is the improved readability, right? So, the only thing you have to practice—those who are learning

Python for the first time—practice the initial code, and then when you come to generics. So, you can find the beauty of this particular interpreter, right?

So, therefore, when you have the improved readability, and this is enforcing generics, but Python remains dynamically typed, right? So, whereas, we had seen in the case of C++, it is compile-time, right? So, since that is the compile-time. So, here it is offering runtime flexibility, right? Python has the runtime flexibility.

And it is easy to use. In fact, when you are writing the code in Python, you can find it very, I mean, easy compared to Java and C++. So, when you are adding this by generics concept, all right? So, in fact, I mean, you can have the flexibility, right? So, whether it is at runtime or you can see the easiness, all right? So, this is all about your templates and. So, in fact, in this chapter, I introduce Python as well, right. So, we have extensively studied C++ and Java, and in fact, we have seen several examples with the help of C++ and Java. So, now, from this chapter, what can you do? You try to write the simple code, right.

So, till, let us say, loop and then see how you have to write, almost the same, all right, but the only thing is the syntax will change for classes and objects. So, once you are familiar, then you start trying to use the concept of generics in Python, right. So, with this, I am concluding this chapter. Thank you all.

Conclusion: Templates vs Generics

- ❏ **C++ Templates:**
 - ❏ Provides compile-time generic programming.
 - ❏ Highly flexible but lacks strict type safety.
 - ❏ Templates can lead to larger binaries due to separate instantiation for each type.
- ❏ **Java Generics:**
 - ❏ Type-safe with compile-time checks but limited by type erasure.
 - ❏ Requires bounded types and wildcards to support complex generic types.
- ❏ **Python Generics with Type Hints:**
 - ❏ Type hints improve readability and enforce generics, but Python remains dynamically typed.
 - ❏ Offers runtime flexibility and ease of use but relies on conventions rather than enforcement.

