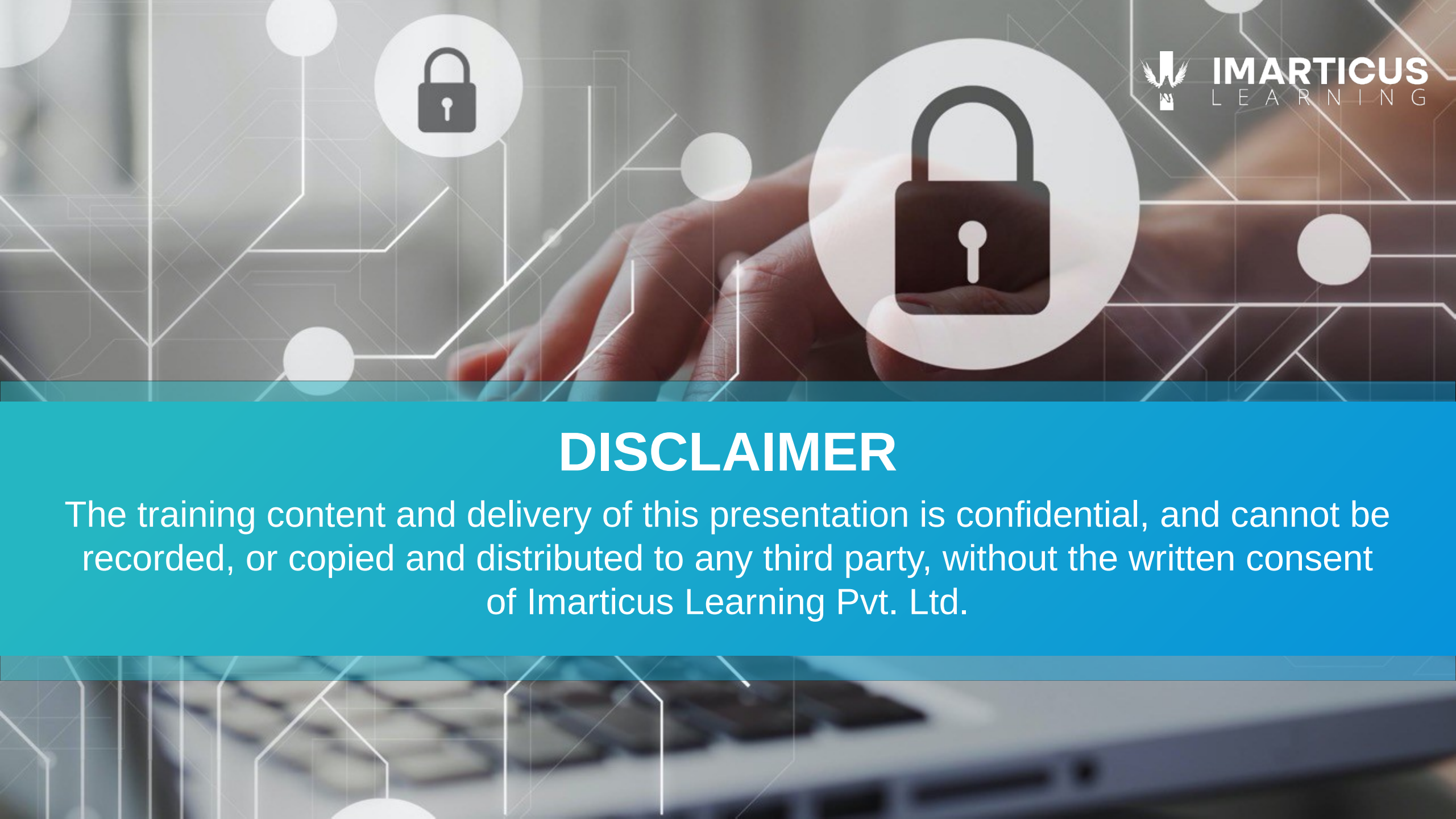
IMARTICUS LEARNING

Python Programming

User-Defined & Lambda functions



IMARTICUS LEARNING

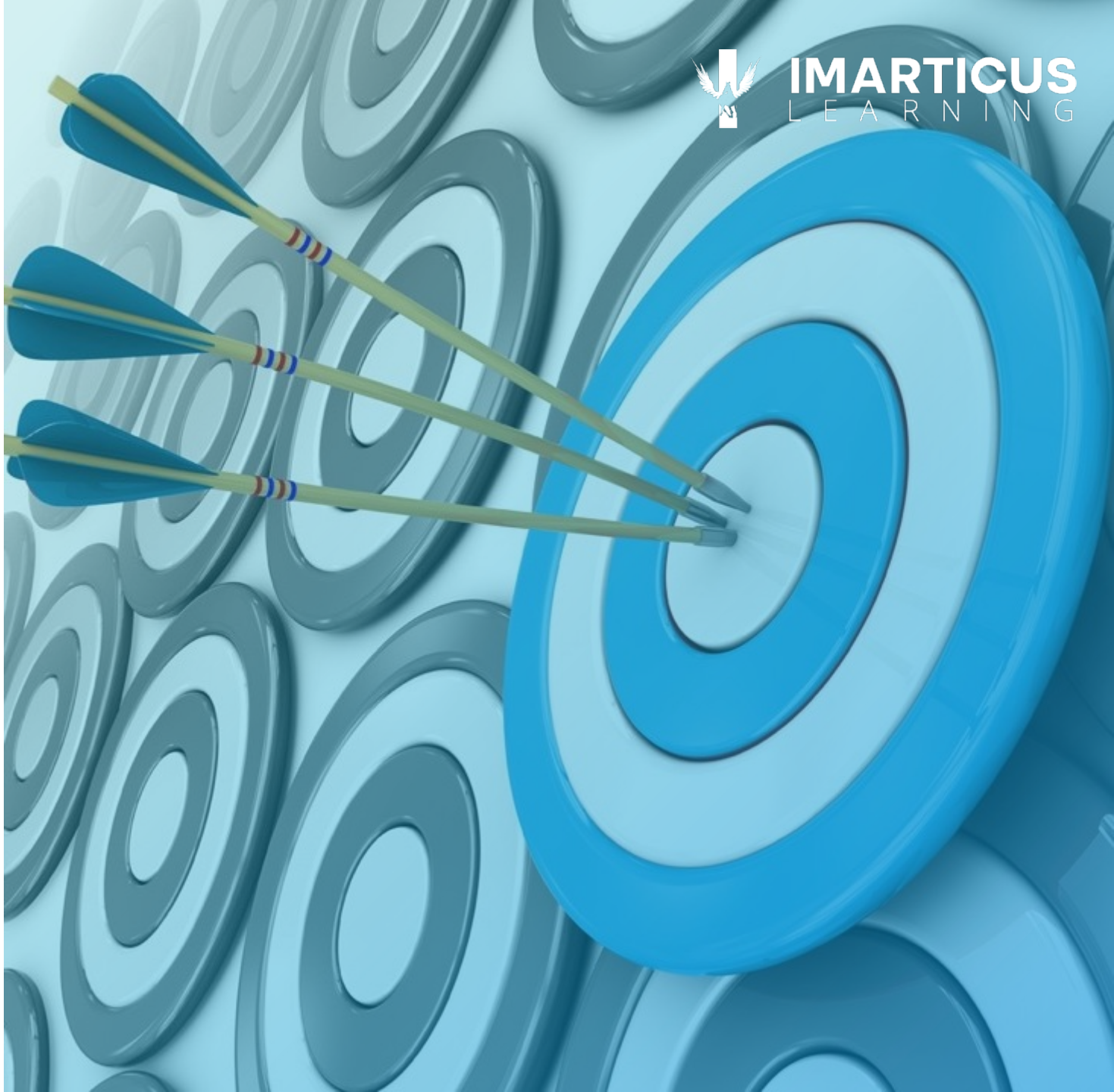
DISCLAIMER

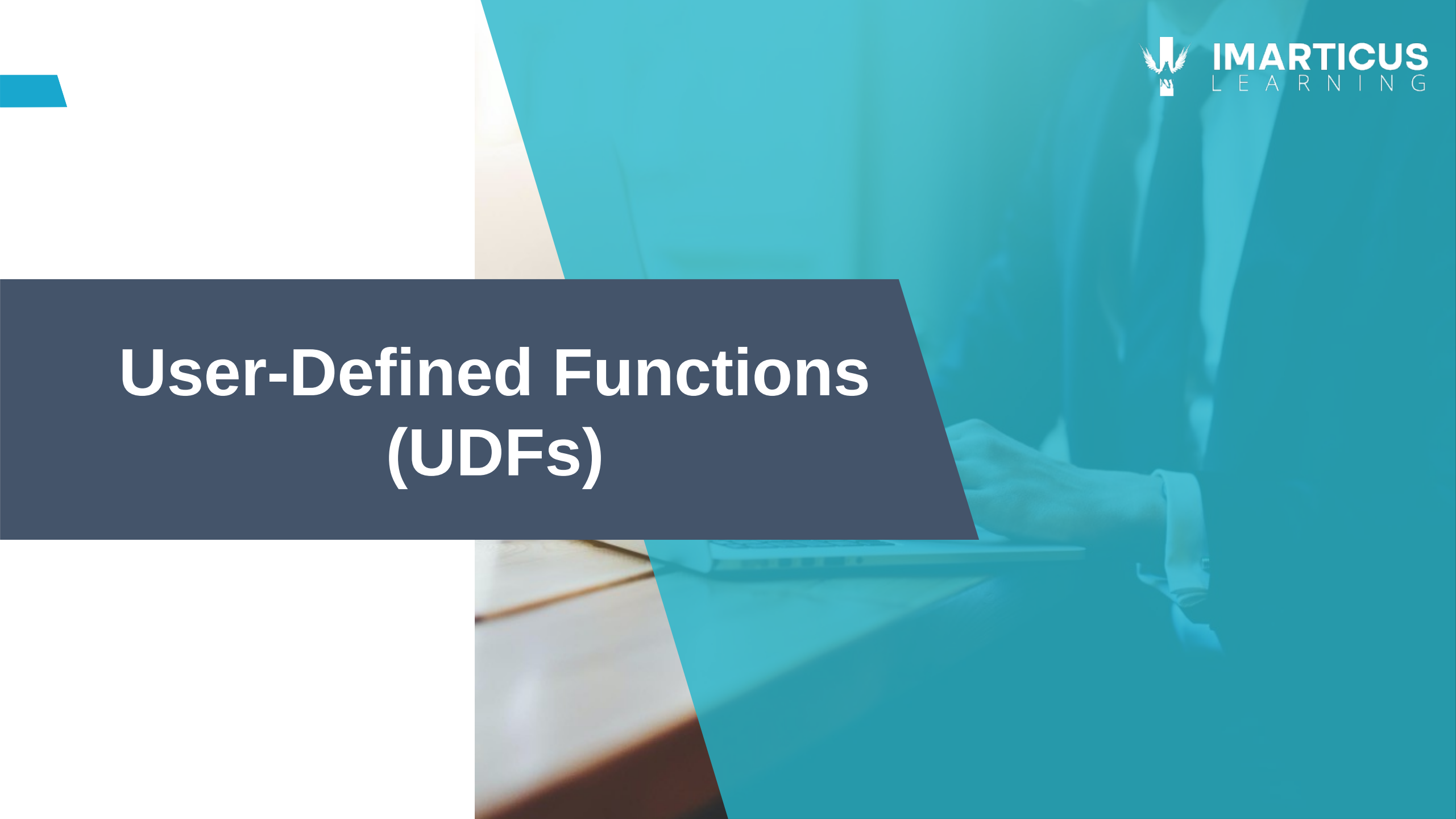
The training content and delivery of this presentation is confidential, and cannot be recorded, or copied and distributed to any third party, without the written consent of Imarticus Learning Pvt. Ltd.

LEARNING OBJECTIVES

At the end of this session, you will learn:

- User-Defined Functions
- Function Arguments
- Lambda Functions





User-Defined Functions (UDFs)

BUILT-IN FUNCTIONS

The functions whose functionalities are pre-defined in Python are called as built-in functions

The Python interpreter has a number of functions built into it that are always available

Some examples of built-in functions are:

- Print()
- Input()
- Min()
- Max()
- Type() etc.

We can defines functions according to our need as well.
Such functions are called as built-in functions.

USER-DEFINED FUNCTIONS

A function that a user defines is known as **user defined function**

A user can give any name to a user-defined function except that the function name should not have any special character including a space character

Any pre-defined Python keywords should not be used as function name

IMARTICUS LEARNING

You cannot use the Python keywords as function name

THE DEF KEYWORD

IMARTICUS LEARNING

- In python, a user-defined function’s declaration begins with the def keyword, followed by the function name
- Keyword **def** marks the start of function header

DEFINING THE USER-DEFINED FUNCTION

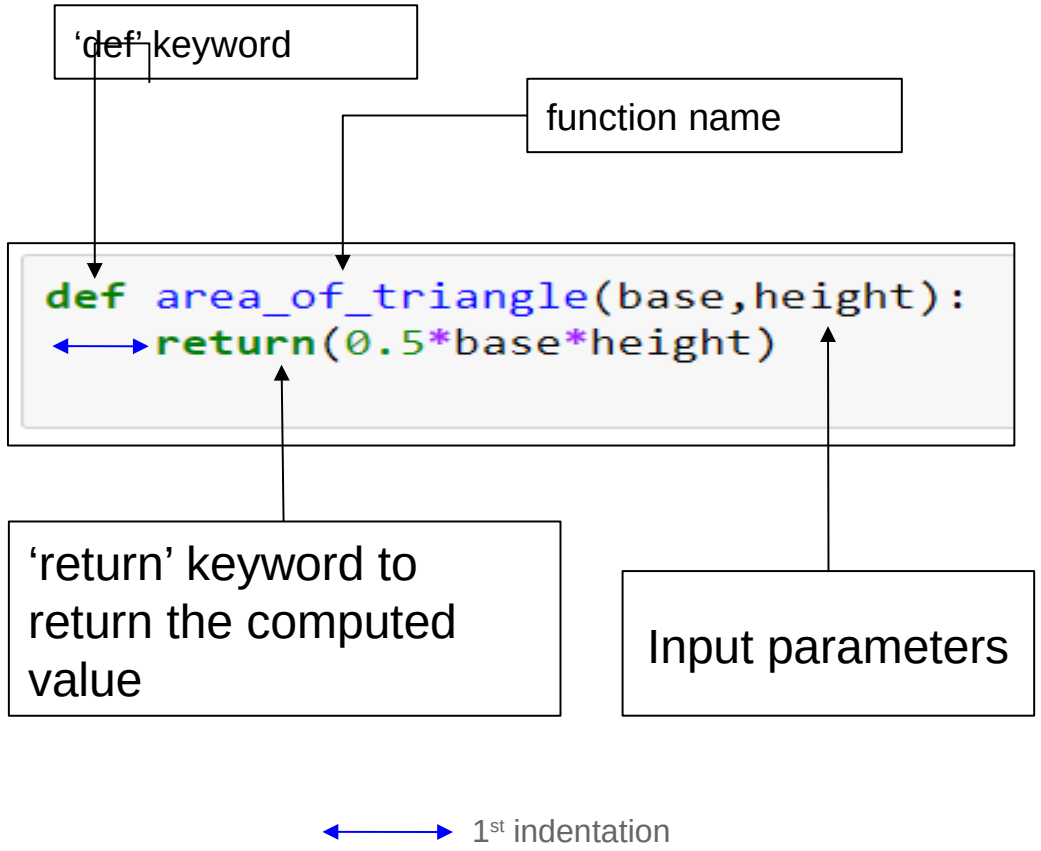
Step 1: Keyword def marks the start of the function header

Step 2: Provide a unique function name

Step 3 (optional): Provide parameters (arguments) by which we pass value to the function

Step 4: Add colon(:) to mark the end of the function header

Step 5: Write the necessary code. A function can be ended with or without a return statement



WRITE YOUR OWN FUNCTION

IMARTICUS LEARNING

```
# create a function 'greet' to display greeting whenever the function is called
def greet():
    print("Hello and welcome")
```

```
# call the function
greet()
```

```
Hello and welcome
```

WRITE YOUR OWN FUNCTION WITH A PARAMETER

IMARTICUS LEARNING

```
# create a function 'greet' that displays a greeting to the person whose name is provided
def greet(name):
    print("Hello", name, "Welcome to the show")
```

```
# take name from the user
person_name = input("Enter your name: ")

# call the function
greet(person_name)
```

```
Enter your name: Shawn
Hello Shawn Welcome to the show
```

IMARTICUS LEARNING

```
# take name from the user
person_name = input("Enter your name: ")

# call the function
greet()
```

```
Enter your name: John

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-3ce897404fa0> in <module>()
      3
      4 # call the function
----> 5 greet()

TypeError: greet() missing 1 required positional argument: 'name'
```

The function will throw an error if we do not pass the required parameter as per the function definition

THE RETURN KEYWORD

IMARTICUS LEARNING

```
# create a function 'addition' that takes two input parameters and adds them
def addition(num1, num2):
    add = num1 + num2
    return(add)
```

```
# call the function by passing two values and assing to a variable
summation = addition(2, 8)
print(summation)
```

```
10
```

IMARTICUS LEARNING

```
# create a function 'addition' that takes two input parameters and adds them
def addition(num1, num2):
    add = num1 + num2
    return
```

```
# call the function by passing two values and assing to a variable
summation = addition(2, 8)
print(summation)
```

```
None
```

The function returns no value if we use return keyword in the function with any return value

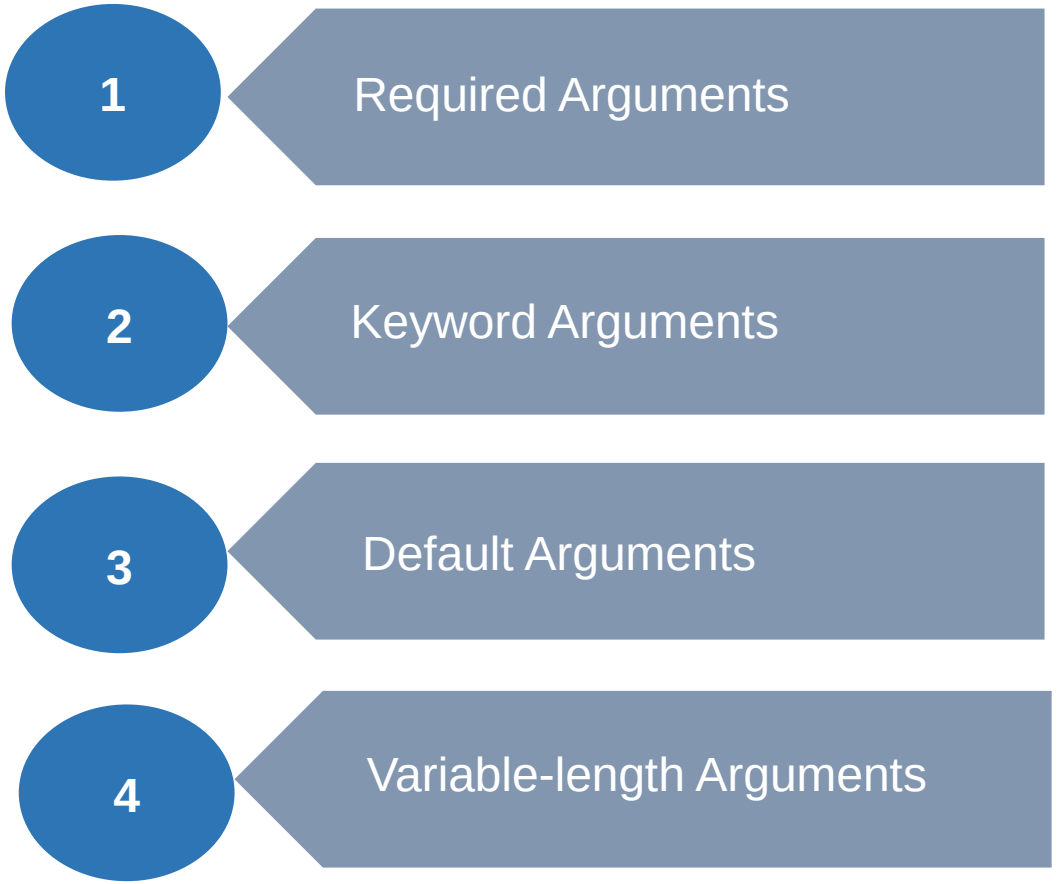
IMARTICUS LEARNING

Function Arguments

FUNCTION ARGUMENTS

IMARTICUS LEARNING

Four types of formal arguments:



REQUIRED ARGUMENTS

IMARTICUS LEARNING

Required arguments are the arguments passed to a function in correct positional order

```
# create a function 'reg_arg' with
def reg_arg(num1, num2):
    print("First argument is: ", num1)
    print("Second argument is: ", num2)
```

```
# call the function by passing two values
num1 = 2
num2 = 8
reg_arg(num2, num1)
```

```
First argument is:  8
Second argument is:  2
```

REQUIRED ARGUMENTS

IMARTICUS LEARNING

In case, if the function does not requires the argument and is still passed it will throw an error

```
# create a function 'greet' that displays a greeting to the person whose name is provided
def greet():
    print("Hello and Welcome to the show")

# take name from the user
person_name = input("Enter your name: ")

# call the function
greet(person_name)
```

```
Enter your name: John

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-c8ce153fb1aa> in <module>()
      3
      4 # call the function
----> 5 greet(person_name)

TypeError: greet() takes 0 positional arguments but 1 was given
```

REQUIRED ARGUMENTS

IMARTICUS LEARNING

In case, if the function requires the argument and is not passed it will throw an error

```
# create a function 'hello' to display greetings for the day
# the function has argument 'name'

def hello(name):
    print("Hello," + name + " have a wonderful day ahead!")
```

```
hello()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-a75d7781aaeb> in <module>
----> 1 hello()

TypeError: hello() missing 1 required positional argument: 'name'
```

```
hello("Steve")
```

```
Hello,Steve have a wonderful day ahead!
```

KEYWORD ARGUMENTS

IMARTICUS LEARNING

- When we call a function with some values, these values get assigned to the arguments according to their position
- When we call functions in this way, the order (position) of the arguments can be changed
- Such arguments are called as keyword arguments

```
# pass the argument and change in position of the argument
def employee(Name, Designation):
    print(Name, Designation)
```

```
# Keyword arguments
employee(Name = "John", Designation="CEO")

employee(Designation = "CEO", Name = "John")
```

```
John CEO
John CEO
```

DEFAULT ARGUMENTS

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument

```
# define a function with a default arguement
def emp(name, sal = 12000):
    print("emp name - ", name)
    print("emp salary - ", sal)

emp("Sam")
```

```
emp name -  Sam
emp salary -  12000
```

We have passed a default value for salary in the function definition

VARIABLE-LENGTH ARGUMENTS

IMARTICUS LEARNING

- Using **args* helps you in passing variable number of arguments
- This is especially helpful when you do not know how many arguments to pass to the function

```
# define a function the returns the addition on numbers passed
def addition(*numbers):
    add = 0
    for num in numbers:
        add = add + num
    print(add)
```

```
addition(1,2)
```

```
3
```

```
addition(5,4,6,7,10)
```

```
32
```

VARIABLE-LENGTH KEYWORDED ARGUMENTS

IMARTICUS LEARNING

- **kwargs allows the users to pass keyword-ed arguments of variable length to a function

```
def employee(**details):
    print(type(details))
    for k, v in details.items():
        print(k, "-->", v)
```

```
employee(first_name = "Jack", salary = 12000)
```

```
<class 'dict'>
first_name --> Jack
salary --> 12000
```

```
employee(first_name = "Jack",seconf_name = "Mayer", salary = 12000, designation = "Financial Advisor")
```

```
<class 'dict'>
first_name --> Jack
seconf_name --> Mayer
salary --> 12000
designation --> Financial Advisor
```

IMARTICUS LEARNING

LAMBDA FUNCTION

LAMBDA FUNCTION

IMARTICUS LEARNING

Lambda functions are anonymous

They do not have any name

‘Lambda’ keyword to be used to create lambda function

Simple one-line function

No **‘def’** or **‘return’** keyword to be used with a lambda function

LAMBDA FUNCTION

IMARTICUS LEARNING

Every lambda function begins with the “lambda” keyword

A lambda function can have multiple arguments separated by commas

A colon precedes the expression

The expression always returns an object

```
lambda arguments : expression
```

LAMBDA FUNCTION

IMARTICUS LEARNING

Python function code to find the greater number

```
def greater(num1,num2):
    if (num1>num2):
        return num1
    else:
        return num2

great = greater(8,15)
print(great)
```

```
15
```

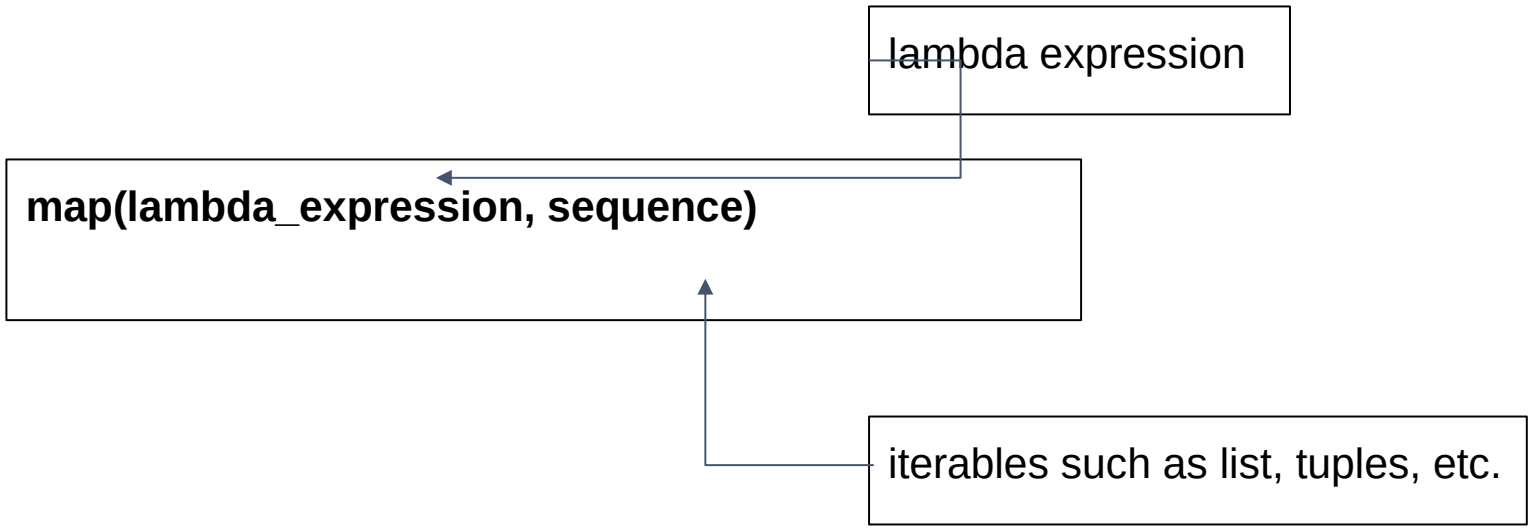
Lambda function to find the greater number

```
greater = lambda num1, num2: num1 if num1 > num2 else num2
greater(8,15)
```

```
15
```

LAMBDA WITH MAP()

- *map()* functions expect a function_object, in our case a lambda function,and an iterable
- It executes the function_object for each element in the sequence and returns a sequence of the elements modified by the function object



LAMBDA WITH MAP()

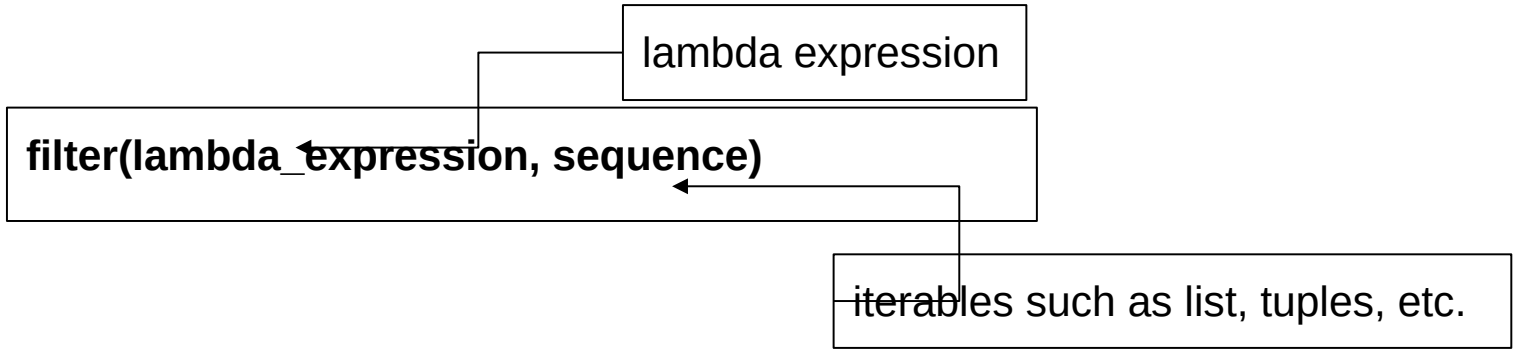
IMARTICUS LEARNING

```
# The output is often type-casted into a seq type, as follow:

num_list = list(range(1,11))
sq_list = list(map(lambda x: x**2, num_list))
sq_list
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

LAMBDA WITH FILTER()

- The *filter()* function expects two arguments: *function_object(lambda)* and an iterable
- *Lambda expression* returns a boolean value and is called for each element of the iterable
- It returns only those elements for which the function_object returns *true*



LAMBDA WITH FILTER()

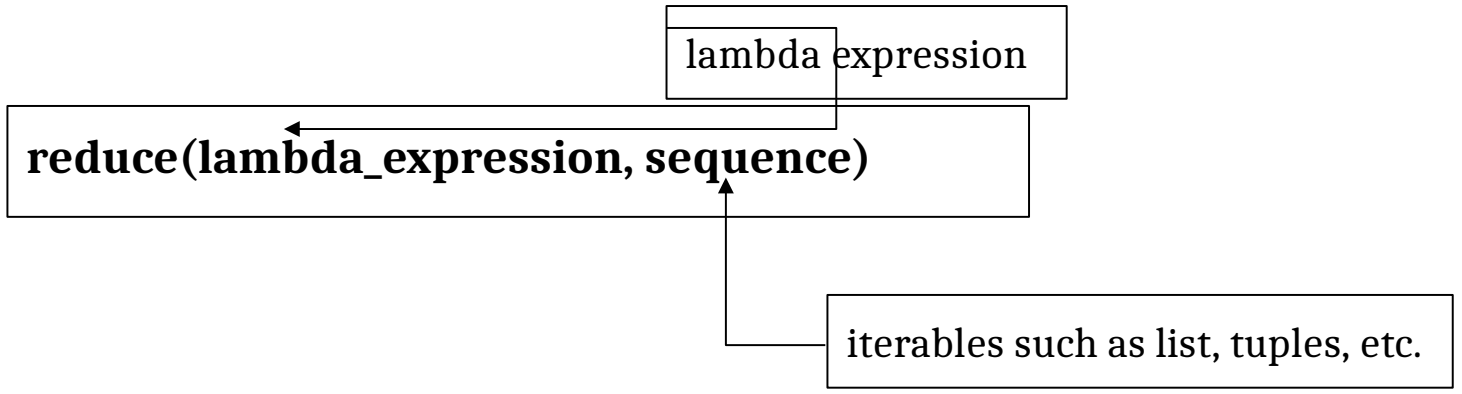
```
# The output is often type-casted into a seq type, as follow:

num_list = list(range(1,11))
sq_list = list(filter(lambda x: x % 5 == 0, num_list))
sq_list
```

```
[5, 10]
```

Unlike map(), the filter() function can have only one iterable as input.

LAMBDA WITH REDUCE()

- The reduce() function in Python takes in a function and a sequence as argument.
- The function is called with a lambda function and a sequence. A new reduced result is returned.
- This performs a repetitive operation over the pairs of the sequence object.



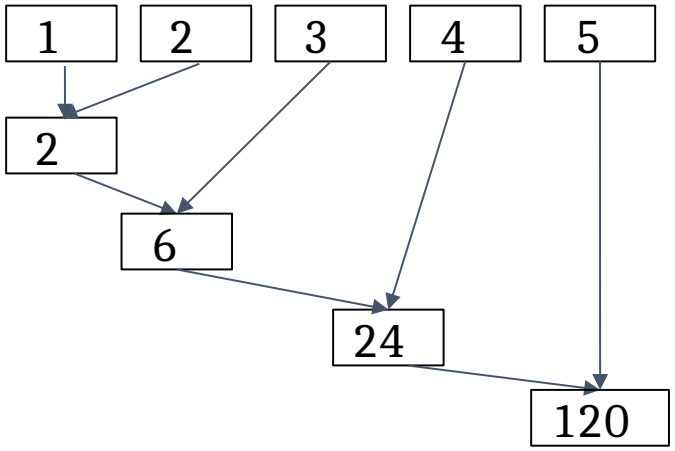
LAMBDA WITH REDUCE()

Determining the factorial of a number

```
# import reduce function from the functools library
from functools import reduce
num_list = [1,2,3,4,5]

# the following function prints factorial of 5
reduce(lambda a,b: a*b, num_list)
```

```
120
```

Working:



LAMBDA WITH REDUCE()

Determining the minimum of a numeric tuple by using reduce:

```
# import reduce function from the functools library
from functools import reduce

# create a numeric tuple
num_tup = (0, -8, 55, -100, 1000, 33)

# the following function prints the minimum value from the iterable passed
reduce(lambda x, y: x if (x<y) else y, num_tup)
```

```
-100
```

Note:

reduce() can only have iterables of same type as input

We're committed to empower you to be

#FutureReady

through powerful training solutions.

IMARTICUS LEARNING

We build the workforce of the future.

250+
Corporate Clients

30,000+
Learners Trained

25000+
Learners Placed

FREE WEBINARS

Sign up for free webinars with industry experts every fortnight!

Enroll Now!

Contact us today!

https://imarticus.org/