Get the best out of Live Sessions

HOW?

e!

Check your Internet Connection

Log in 10 mins before and check your internet connection to avoid any network issues during the LIVE session.

Speak with the Instructor

By default, you will be on mute to avoid any background noise. However, if required you will be **unmuted by instructor**.

Clear Your Doubts

Feel free to clear your doubts. Use the **Questions** tab on your webinar tool to interact with the instructor at any point during the class.

Let Us Know If You Liked Our Content

Please share feedback after each class. It will help us to enhance your learning experience.

edureka!

Python Programming Certification Training

COURSE OUTLINE
MODULE NO

1. Introduction to Python
2. Sequences and File Operations
3. Deep Dive Functions and OOPs
4. Working with Modules and Handling Exceptions
5. Introduction to NumPy
6. Data Manipulation Using Pandas
7. Data Visualization Using Matplotlib
8. GUI Programming

edureka!

Module 1: Introduction to Python

Titles

- Use Case – Need of Programming
- Advantages of programming
- Demand for Python
- Application of Python in Different Domains
- Fundamentals of Python
- Using the Print Statement
- Standard Data Types
- Python Operators
- Control Structures: Conditional Statements
- Control Structures: Loops
- Structural Pattern Matching

Learning Objective(s)

By the end of this module, you will be able to:

- Explain the need for programming
- Elaborate on Python and its basic
- Discuss why to choose Python over other languages
- List various applications of Python in different domains
- Implement various Python concepts – variables and data types
- Use operators, conditional statements, and loops
- Analyze flow control

Use Case Example of Python

edureka!

Course Management System: Use Case

John's job is to film and edit courses for a website.

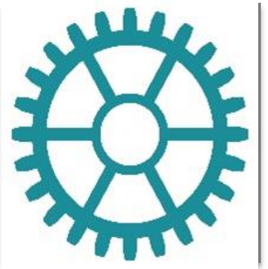
John

Course Management System (contd.)

In the process of preparing a course, John needs to deal with many different files.

Video Files

Audio Files



Motion Graphics

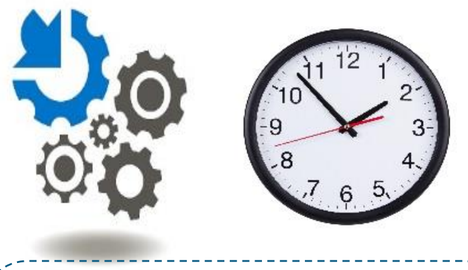
He creates a set of folders to organize these materials by course, lesson, and type of file.

Physics 

Chemistry 

Maths 

Course Management System: Problem



John wants a proper methodology to automate this task and save a lot of time.

Creating these folders takes a lot of time

He was creating these folders manually

Physics

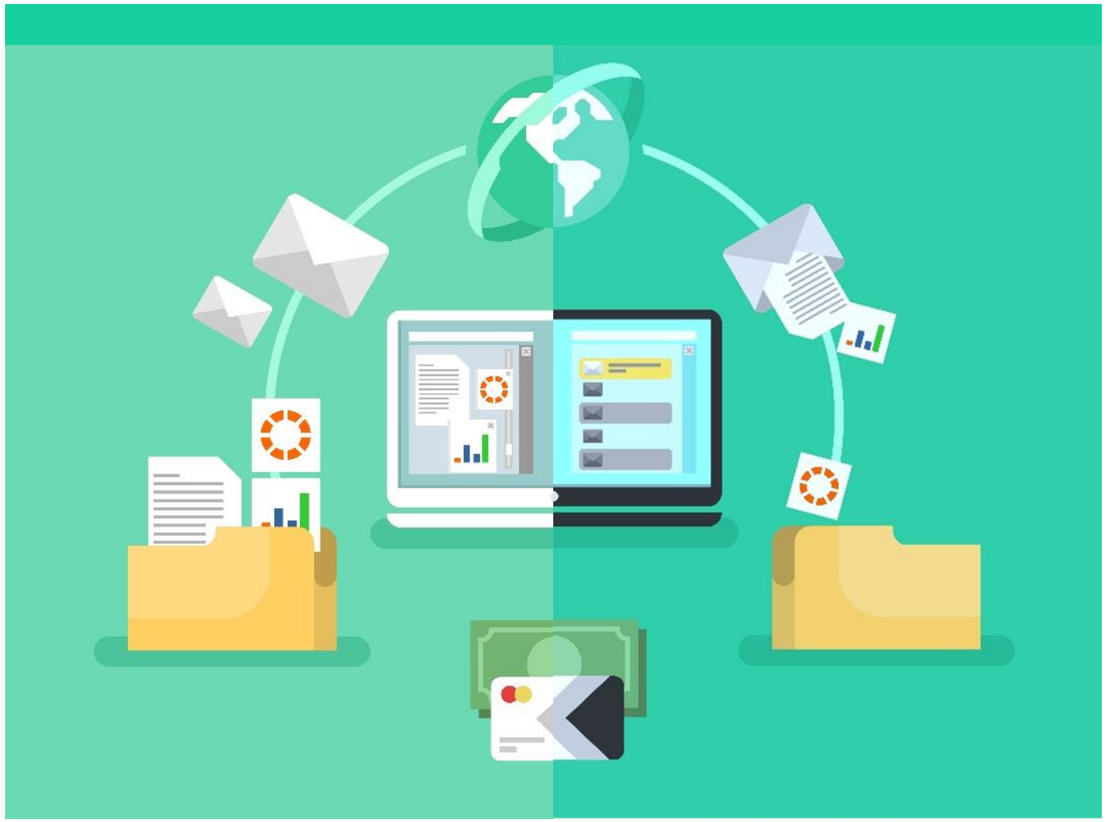
Chemistry

Maths

Course Management System: Solution

John learned to program and designed a system, which automatically creates folders, the name of the course, the number of lessons at the end of the course, and more.

John



Advantages of Programming

edureka!

Advantages of Programming

- Enables to develop of applications and games
- Keeps track of historic data and analyzes it
- Provides better experience to end users
- Automates time consuming tasks
- Helps in streamlining of processes
- Algorithms can be easily customized

Different Programming Languages

C++

Matlab

Kotlin

Python

Scala

Java

PHP

Features of Python

Simplicity

Python is a beginner-friendly language with neat and lucid syntax.

Free and open-source

Python is free and open-source software, that is, one can freely distribute copies of this software, read its source code and modify it.

Supports different programming paradigms

Supports procedure-oriented programming as well as object-oriented programming.

Procedure Oriented

Object Oriented

python

Features of Python (contd.)

Platform independent

Python programs written in one operating system can be executed on another OS without any modifications.

Powerful and open-source IDEs

Wide range of open-source and powerful IDEs for practicing python

Libraries: there's one for every need!

Python has a huge range of data analytics libraries, such as sklearn, tensorflow, keras, pandas, NumPy, Matplotlib, and so on.

Demand of Python

edureka!

Tech Giants Using Python

The popular YouTube video sharing system is largely written in Python

Google makes extensive use of Python in its web search system

Dropbox storage service codes both its server and client software primarily in Python

The Raspberry Pi single-board computer promotes Python as its educational language

Organizations Using Python

BitTorrent peer-to-peer file sharing system began its life as a Python Program

NASA uses Python for specific Programming Task

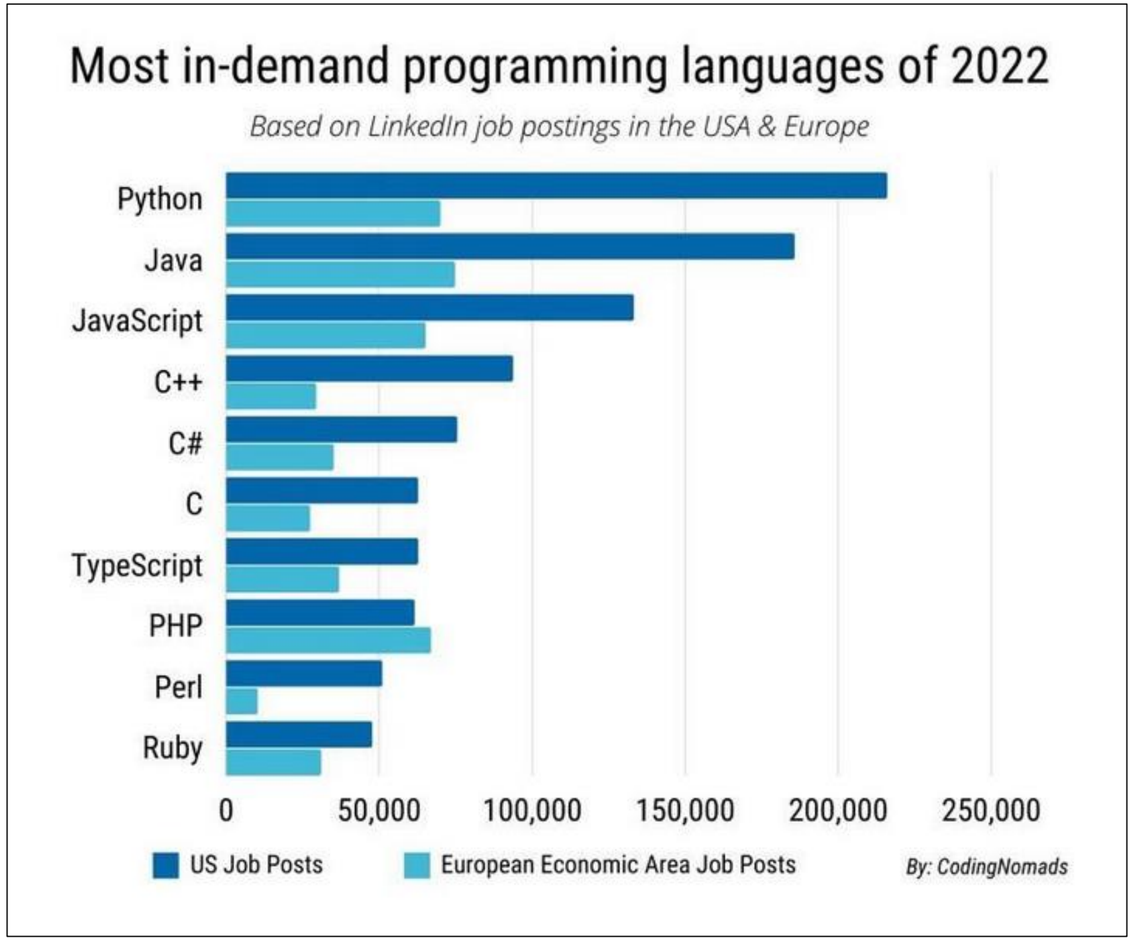
The NSA uses Python for cryptography and intelligence analysis

Netflix and Yelp have both documented the role of Python in their software infrastructures

Applications of Python

Python Job Trends

Python is one of the Top 3 Programming Languages of 2022.



Fundamentals of Python

edureka!

Python Program Example

Code example

```
#Take input from user
num = float(input('Enter a number: '))

'''use the user input
   to calculate square root of the number'''

num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))

if(num_sqrt <=1):
    print ('Foo')
else:
    print('Bar')
```

Code output

```
Enter a number: 0.25
The square root of 0.250 is 0.500
Foo
```

Python Code Execution

Source Code

The structured code is written and saved with .py extension.

Byte Code

The code is converted into Byte code for machine to understand.

PVM

The Byte codes are executed.

Comments and Literals

Comments: Any text to the right of the # symbol is mainly used as notes for the readers. Statements on the right side of # do not get executed. It gives more information about the function.

Bulk comments: Enclose the code in triple quoted strings (“”””).

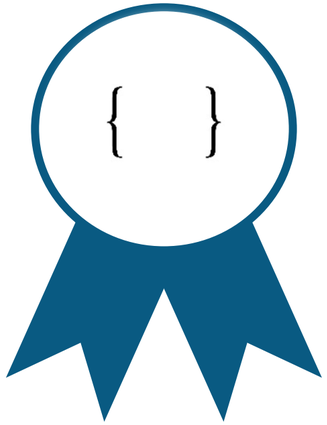
Literal constants: Any number or character or set of characters.

Indentation: It refers to the spaces at the beginning of a code line. Python uses indentation to indicate a block of code.

```
#Take input from user
num = float(input('Enter a number: '))

'''use the user input
   to calculate square root of the number'''

num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))

if(num_sqrt <=1):
    print ('Foo')
else:
    print('Bar')
```

edureka!

Indentation



No braces to indicate blocks of code for class and function definitions or flow control.

Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

Leading whitespace at the beginning of a logical line is used to compute the indentation level of the line, which in turn, is used to determine the grouping of statements.

Identifier

- A Python Identifier is a name used to identify a variable, function, class, module, or other objects.
- An identifier starts with a letter (A to Z or a to z) or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
- Python is case-sensitive.
- Python does not allow special characters such as @, $, and % within identifiers.

```
#Take input from user
num = float(input('Enter a number: '))

'''use the user input
   to calculate square root of the number'''

num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))

if(num_sqrt <=1):
    print ('Foo')
else:
    print('Bar')
```

Keywords

- These are reserved words having special meaning, and you cannot use them as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

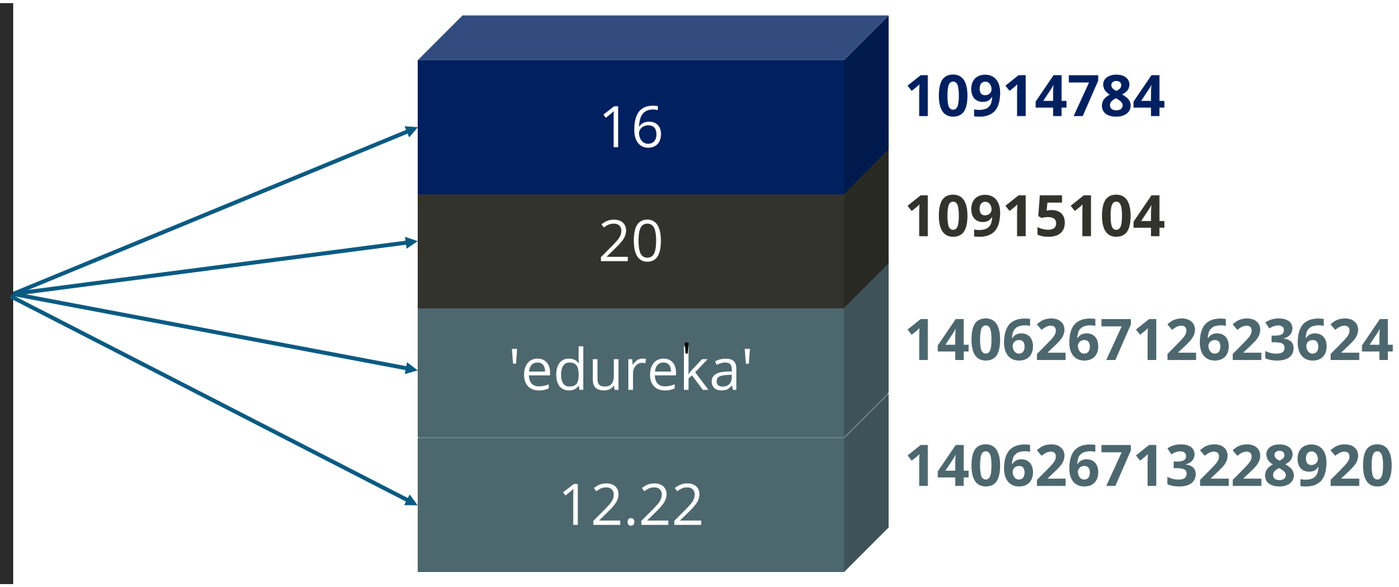
```
#Take input from user
num = float(input('Enter a number: '))

'''use the user input
   to calculate square root of the number'''

num_sqrt = num ** 0.5
print('The square root of %0.3f is %0.3f'%(num ,num_sqrt))

if(num_sqrt <=1):
    print ('Foo')
else:
    print('Bar')
```

edureka!

Variables

Variables are reserved memory locations to store values. This means that when you create a variable, you reserve some space in the memory.

```
a = 16
id(a)
b = 20
id(b)
b ='edureka’
id(b)
b = 12.22
id(b)
```



Note: `id()` is a python inbuilt function that returns the unique identity of an object.

edureka!

Variables (contd.)

Consider the below code, it explains how to assign a value to a **variable.**

Assigning values 10 and edureka! to variables A and B respectively.

Code example

```
A=10
B='edureka!'
print(A,B)
```

Code output

```
10 edureka
```

Using the Print Statement

edureka!

Python Print Statement

`print()` - prints the message to the screen or any other standard output device.

Code example

```
# This line will automatically add a new line before the
# next print statement
print ("Edureka is the best platform for Python content")

# This print() function ends with "**" as set in the end argument.
print ("Edureka is the best platform for Python content", end= "**")
print("Welcome to Edureka!!!")
```

Code output

```
Edureka is the best platform for Python content
Edureka is the best platform for Python content**Welcome to Edureka!!!
```

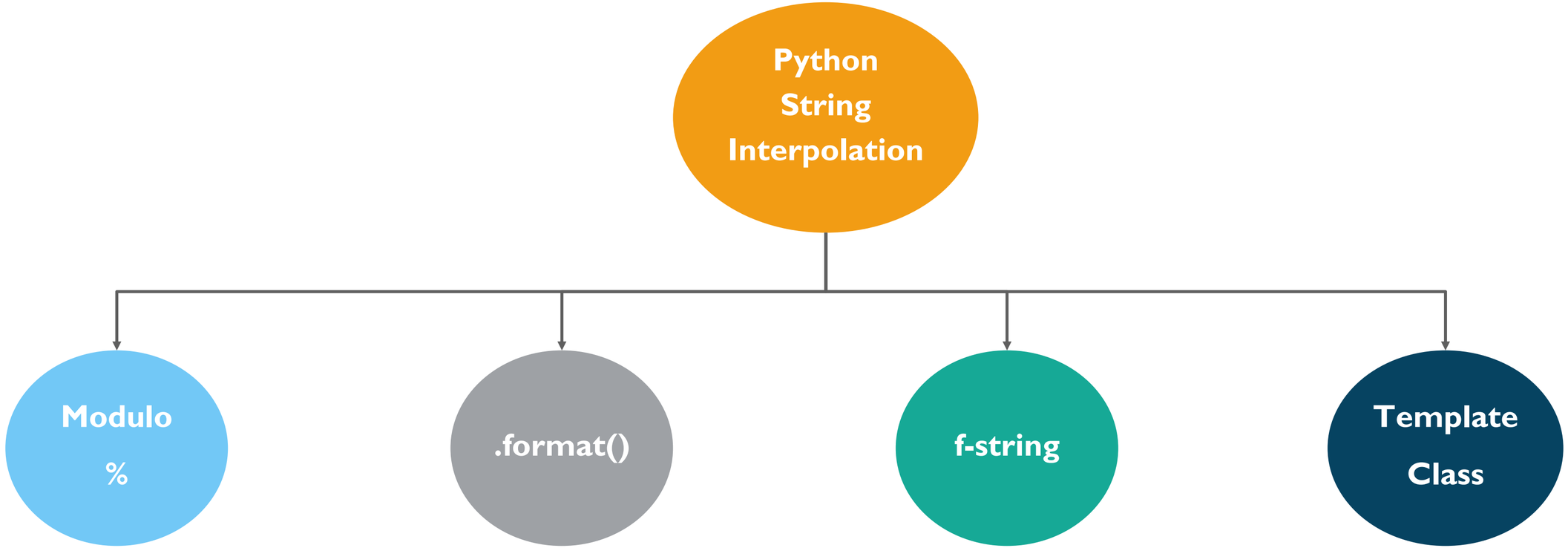
Now let us see how to use formatting in `print()`

edureka!

String Interpolation

String Interpolation is the process of substituting values of variables into placeholders in a string

The string interpolation methods:



edureka!

Modulo Formatting

It is a feature provided by Python which can be accessed with a % operator. This is like printf style function in C

Code example

```
# Python program to demonstrate
# string interpolation


n1 = 'Hello'
n2 = 'Edureka'

# for single substitution
print("Welcome to % s" % n2)

# for single and multiple substitutions()
# mandatory
print("% s ! This is % s." % (n1, n2))
```

Code output

```
Welcome to Edureka
Hello ! This is Edureka.
```

edureka!

str.format()

It works by putting in one or more replacement fields and placeholders defined by a pair of curly braces { } into a string. The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.

Code example

```
# Python program to demonstrate
# string interpolation


n1 = 'Hello'
n2 = 'Edurekans'

# for single substitution
print('{}, {}'.format(n1, n2))
```

Code output

```
Hello, Edurekans
```

f-strings (String Literal Interpolation)

To create an f-string, prefix the string with the letter " f ". The string itself can be formatted in much the same way that you would with `str.format()`.

f-strings provide a concise and convenient way to embed python expressions inside string literals for formatting.

Code example

```
# Python program to demonstrate
# string interpolation


n1 = 'Hello'
n2 = 'Edureka'

# f tells Python to restore the value of two
# string variable name and program inside braces {}
print(f"{n1}! This is {n2}")
```

Code output

```
Hello! This is Edureka
```

Template Class

In the string module, Template Class allows us to create simplified syntax for output specification.

- The format uses placeholder names formed by $ with valid Python identifiers (alphanumeric characters and underscores).
- Surrounding the placeholder with braces allows it to be followed by more alphanumeric letters with no intervening spaces.
- Writing $$ creates a single escaped $.

Code example

```
# Python program to demonstrate
# string interpolation


from string import Template

n1 = 'Hello'
n2 = 'Edureka'

# made a template which we used to
# pass two variable so n3 and n4
# formal and n1 and n2 actual
n = Template('$n3 ! This is $n4.')

# and pass the parameters into the template string.
print(n.substitute(n3=n1, n4=n2))
```

Code output

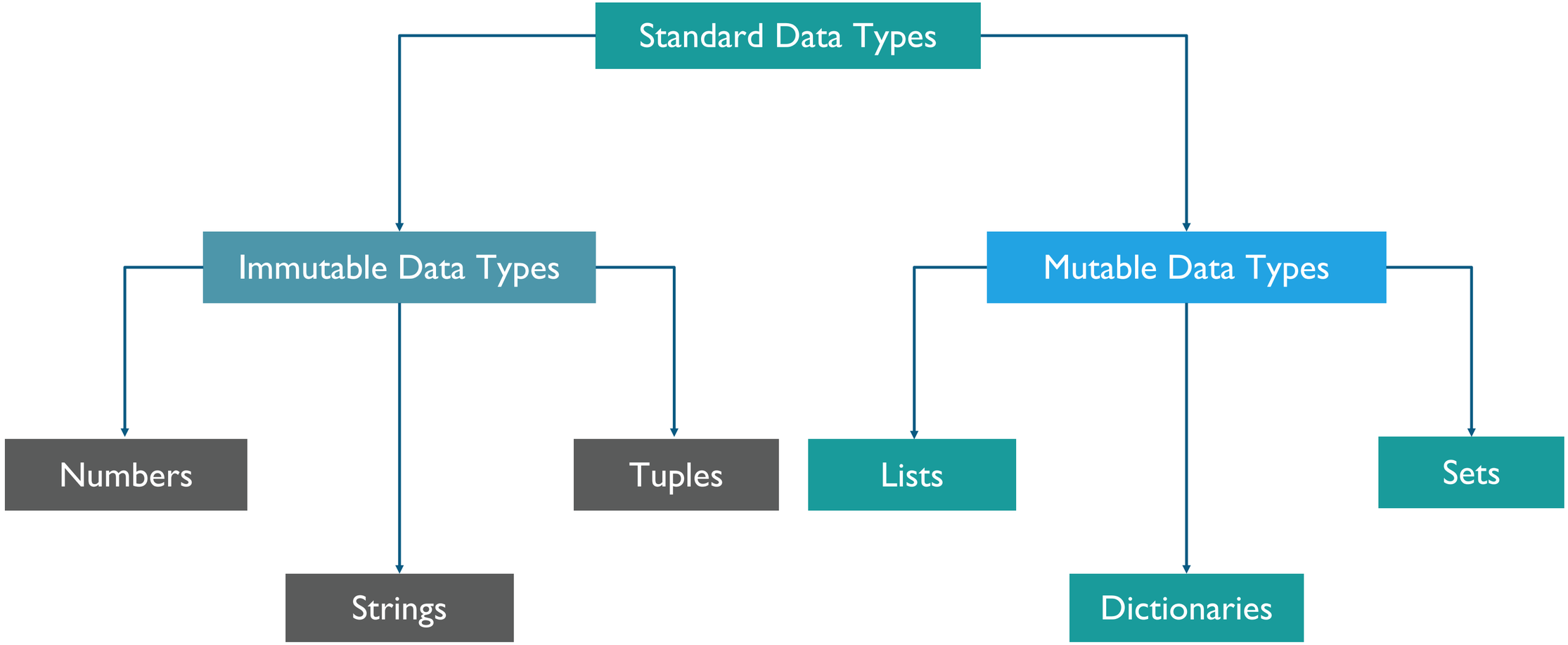
```
Hello ! This is Edureka.
```

edureka!

Standard Data Types

edureka!

Data Types in Python



edureka!

Immutable and Mutable Data Types

```
In [47]: lang = 'Java'

In [48]: print('Original ID of lang variable -->',id(lang))
         Original ID of lang variable --> 1693849167280

In [49]: lang = 'Python'

In [50]: print('ID after assigning new value Python to lang variable -->',id(lang))
         ID after assigning new value Python to lang variable --> 1693841712112
```

Values of immutable objects cannot be changed.

Note: Memory address (ID) is changed.

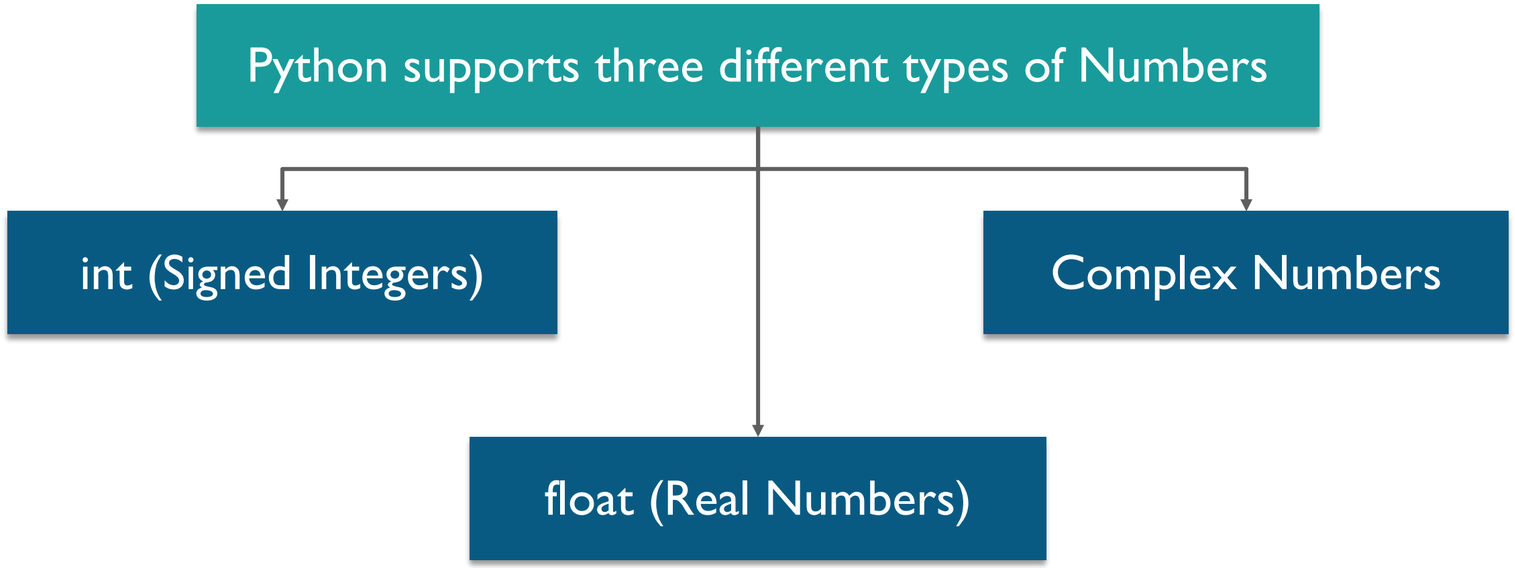
```
In [51]: lst = [1,2,3,4]

In [53]: print('Original ID of the list -->',id(lst))
         Original ID of the list --> 1693923158920

In [54]: lst.append(5)

In [55]: lst
Out[55]: [1, 2, 3, 4, 5]

In [56]: print('ID of the list after appending 5 -->',id(lst))
         ID of the list after appending 5 --> 1693923158920
```

Values of Mutable Objects can be changed.

Note: Memory address (ID) remains the same.

edureka!

Immutable Data Types: Numbers



In Python you can represent **Numbers** in multiple ways:

- Binary
- Octal
- Hexadecimal

Immutable Data Types: Numbers (contd.)

Numbers

Strings

Tuples

```
A = 10

B = 10.65

C = 10+6j

print(A,B,C)
```

```
10 10.65 (10+6j)
```

Integer

Float

Complex

Code output

Numbers in Python: Type and Instance

We can check the type of variable or the value or the function to which class it belongs using `type()` and `isinstance()`.

Code example

```
x=10
y=22.33
z=44+55j


#Type
print(type(x))
print(type(y))
print(type(z))

#Check instance
print(isinstance(x,int))
print(isinstance(x,float))
```

Code output

```
<class 'int'>
<class 'float'>
<class 'complex'>
True
False
```

edureka!

Numbers in Python (Demonstration)

Note: Refer to Module 1: Demo 1 file on LMS for detailed steps.

edureka!

Immutable Data Types: Strings

Numbers

Strings

Tuples

- **Strings** are a sequence of characters represented within quotes.
- Characters in python are treated as strings of length one.

```
A = 'Welcome To edureka!’
B = "Python is Great"
C = ''' High
      level '''
D = """Programming
      Language"""
print(A,B,C,D)
```

Different ways of defining a string

```
Welcome To edureka! Python is Great High
level Programming
Language
```

Code output

edureka!

Immutable Data Types: Tuples

Numbers

Strings

Tuples

A tuple is a collection of objects separated by commas enclosed within parenthesis.

Code example

```
A=(1,2,3.15,'edureka!’)

print(A)
```

A tuple can have objects of different data types, unlike arrays in **‘C’**.

Code output

```
(1, 2, 3.15, 'edureka!')
```

edureka!

Mutable Data Types: Lists

Lists

Dictionaries

Sets

List is a collection of objects separated by comma enclosed within square brackets.

Code example

```
A=[1,2,3.15,'edureka!']

print(A)
```

Lists are enclosed within square brackets.

Code output

```
[1, 2, 3.15, 'edureka!']
```

edureka!

Immutable Data Types: Dictionaries

Lists

Dictionaries

Sets

- Dictionaries are a collection of objects as a key-value pair.
- Each key and its value is separated by a colon (:).
- The key-value pair is separated by commas.
- All the key-value pairs are enclosed within curly braces.

Keys

```
A={'Age':24 ,'Name':'John' }

print(A)
```

Values

Code example

Code output

```
{'Age': 24, 'Name': 'John'}
```

edureka!

Mutable Data Types: Sets

Lists

Dictionaries

Sets

Collection of unordered unique items inside curly braces {}, separated by a comma.

Every element in a set must be unique.

```
A={1,2,3,3}

print(A)
```

Code example

Notice that 3 has appeared only once

```
{1, 2, 3}
```

Code output

You can also create a set by calling a built-in function `set()`

edureka!

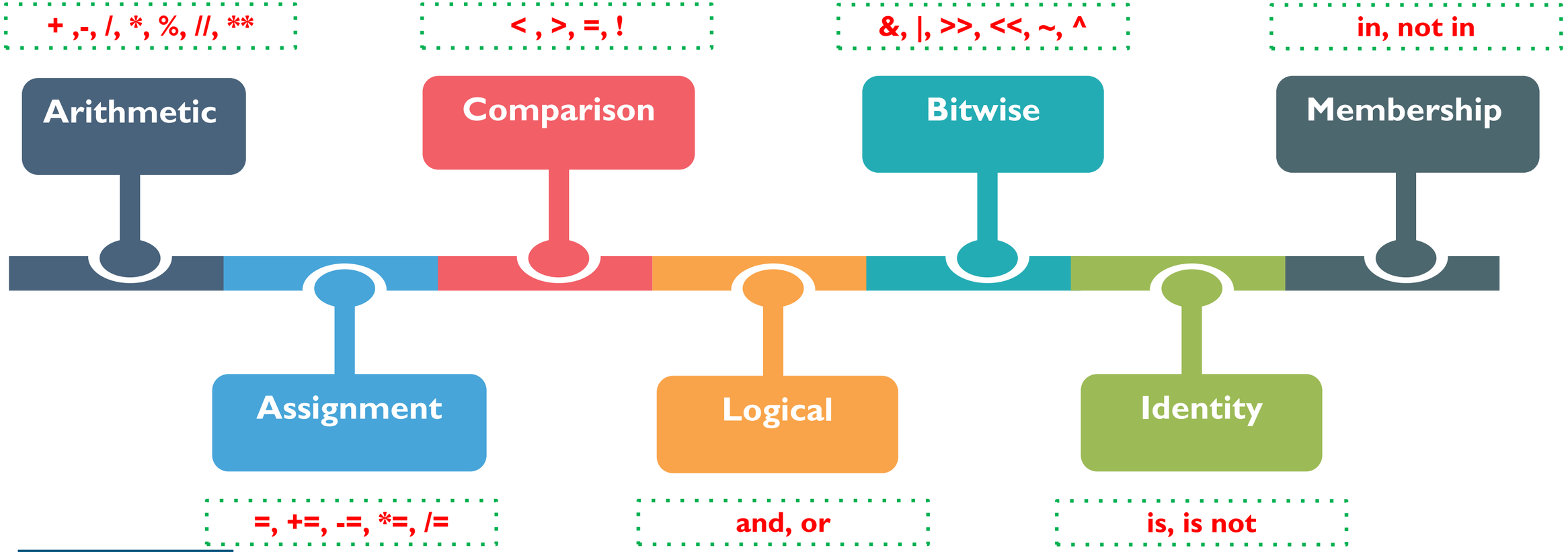
Introduction to Operators in Python

edureka!

Operators in Python

Operators perform operations on the Operands.

Example: 2 + 3 = 5, here **2** and **3** are **Operands** and **+** is the **Operator**.



edureka!

Operator Precedence

Precedence is the order in which the operations are carried out.

The operator precedence in Python is listed in the following table. It is in descending order (the upper group has higher precedence than the lower ones).

Operators	Meaning
`()`	Parentheses
`**`	Exponent
`+x`, `-x`, `~x`	Unary plus, Unary minus, Bitwise NOT
`*`, `/`, `//`, `%`	Multiplication, Division, Floor division, Modulus
`+`, `-`	Addition, Subtraction
`<<`, `>>`	Bitwise shift operators
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==`, `!=`, `>`, `>=`, `<`, `<=`, `is`, `is not`, `in`, `not in`	Comparisons, Identity, Membership operators

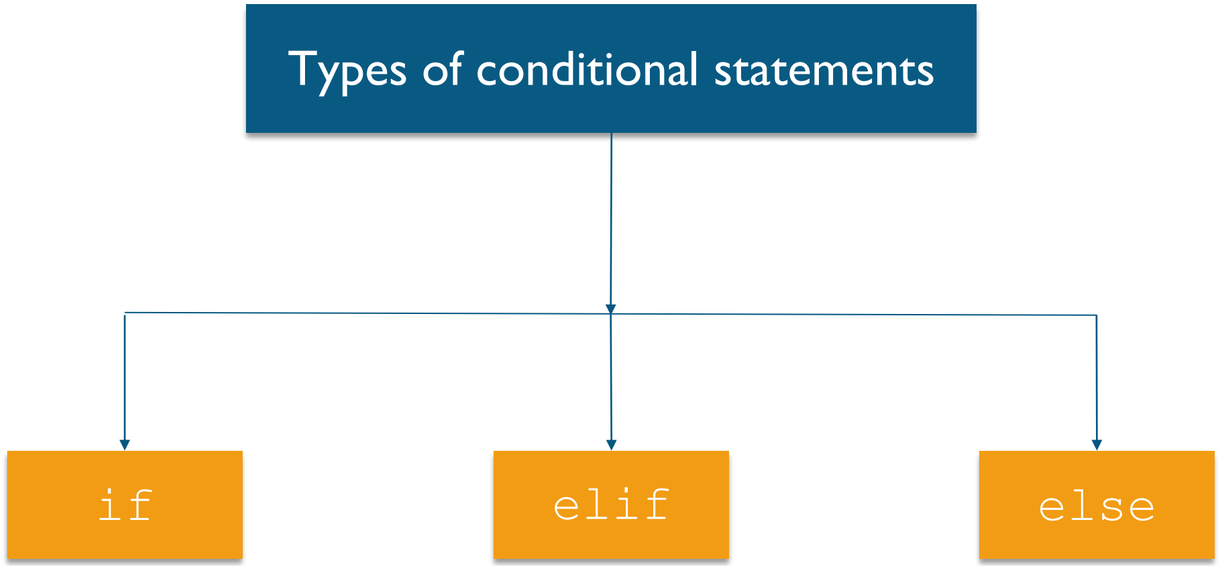
edureka!

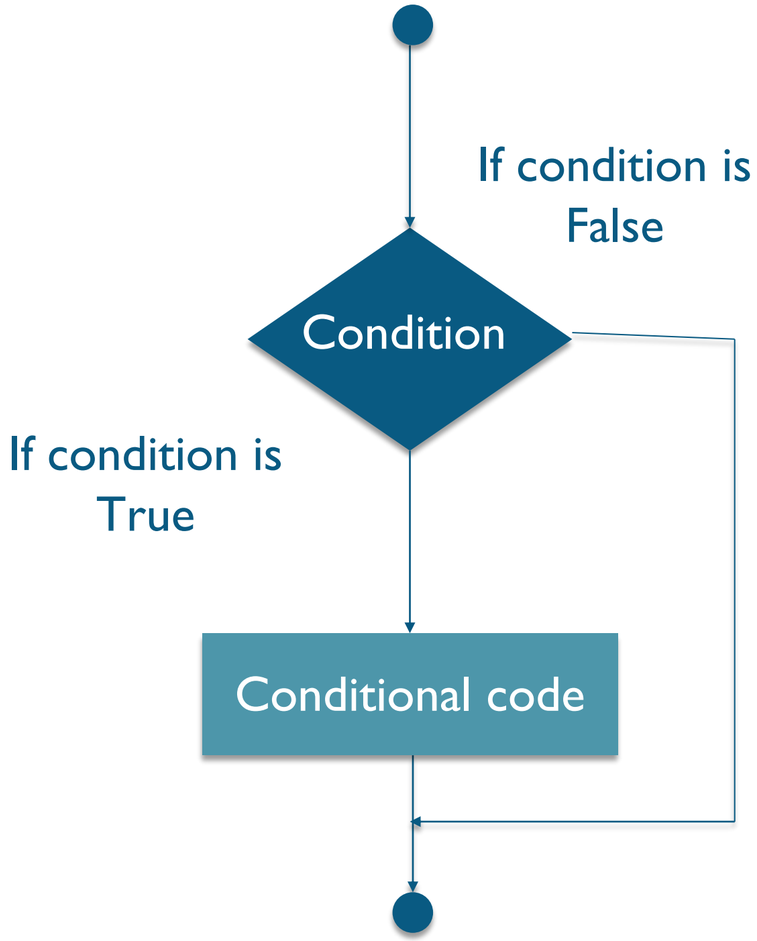
Control Structures: Conditional Statements

edureka!

Conditional Statements

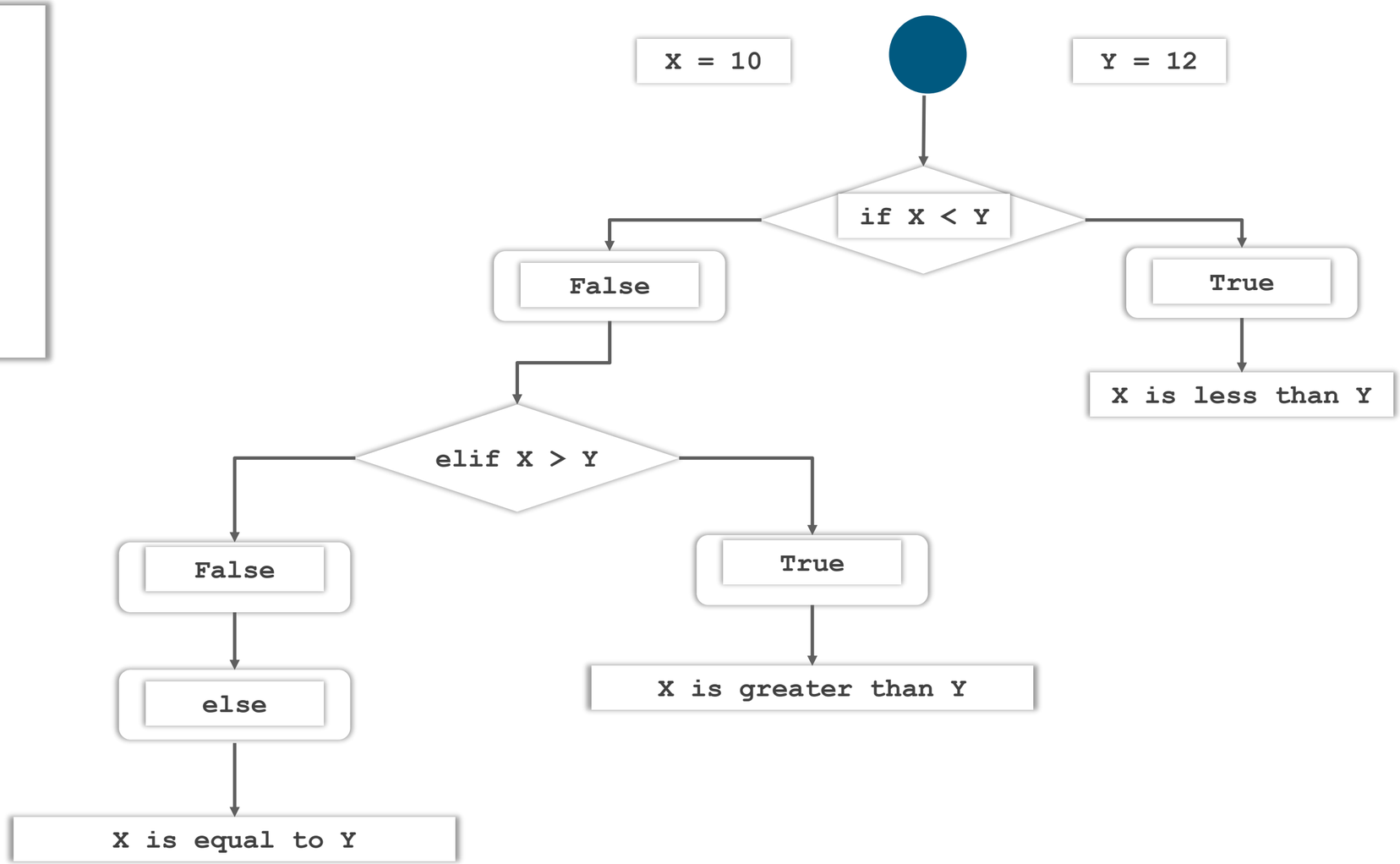
Conditional statements are used to execute a statement or a group of statements when some condition is **True.**





Conditional Statements: `if`, `elif`, and `else`

```
X = 10
Y = 12
if X<Y:
    print('X is less than Y')
elif X>Y:
    print('X is greater than
Y')
else:
    print('X is equal to Y')
```

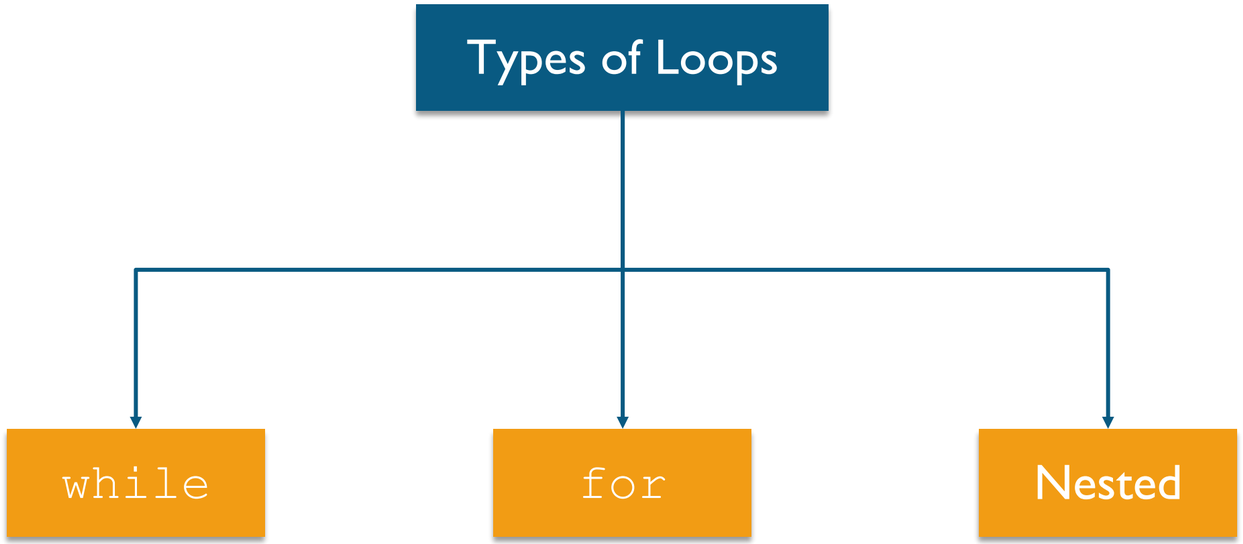


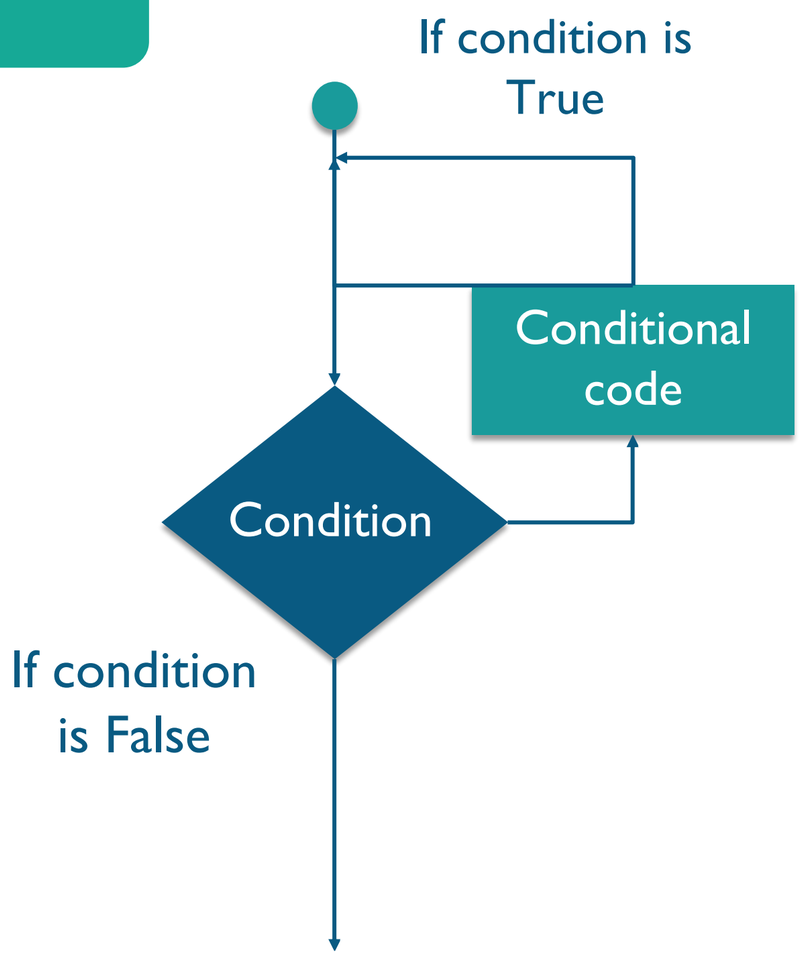
Control Structures: Loops

edureka!

Loops

A loop statement allows us to execute a statement or a group of statements multiple times.





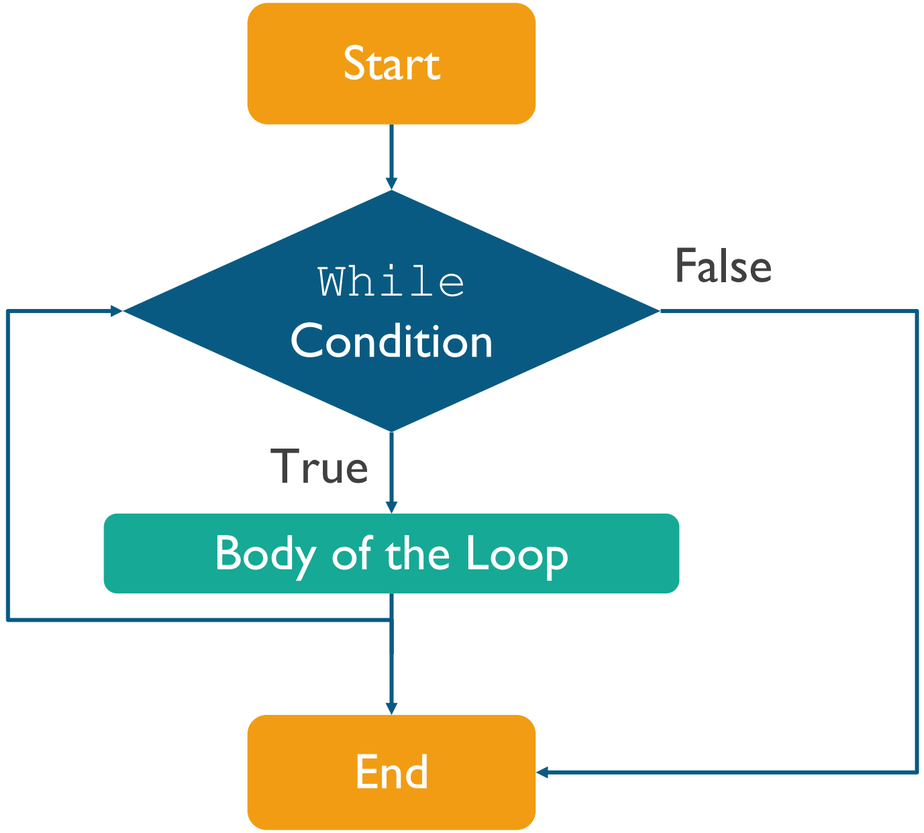
edureka!

while Loop

`while` loops keep on iterating the statement/block until certain conditions are met.

Syntax

```
while expression:
    statements
```



while Loop (contd.)

Code example

```
count=0
while(count<5):
    print(count)
    count=count+1

print("Good bye!")
```

Condition

Conditional code

Code output

```
0
1
2
3
4
Good bye!
```

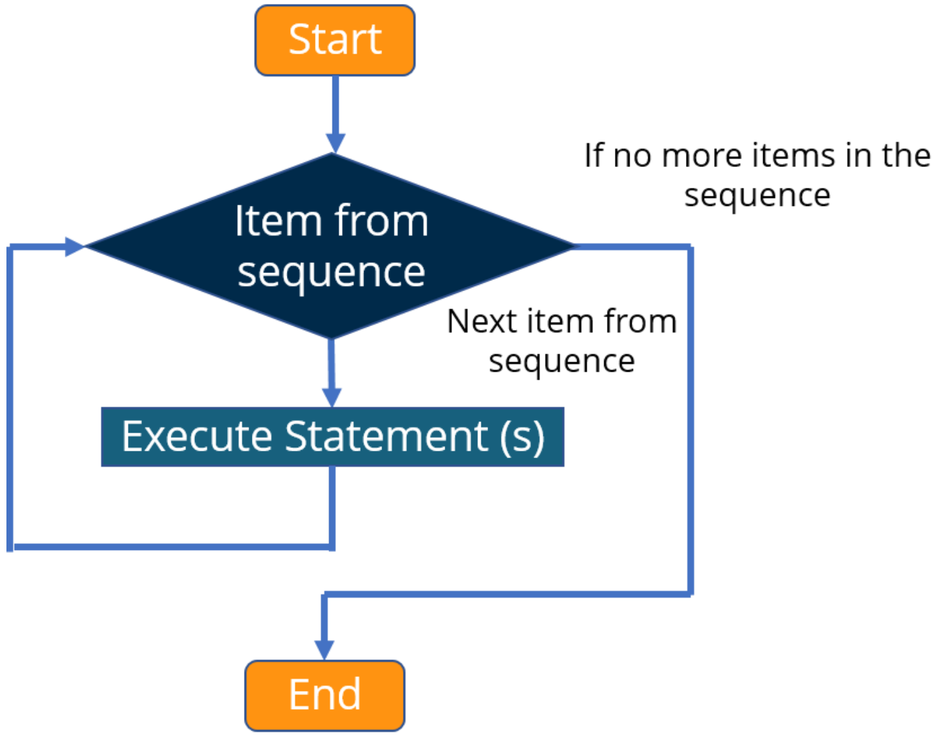
Prints all the integers between 0 and 5

for Loop

`for` is a conditional iterative statement used to execute the block for a specified number of times(based on the elements in range or sequence).

Syntax

```
for <variable> in <range>:
    stmt1
    stmt2
    ...
    stmtn
```



for Loop (contd.)

Please note that, the stop index is not included in the result of *for* loop

Code example

```
for i in range(1,5):
    print("Welcome to for loop",i)
```

Code output

```
Welcome to for loop 1
Welcome to for loop 2
Welcome to for loop 3
Welcome to for loop 4
```

Code example

```
for i in range(5,1,-1):
    print("Welcome to for loop",i)
```

Code output

```
Welcome to for loop 5
Welcome to for loop 4
Welcome to for loop 3
Welcome to for loop 2
```

edureka!

for-else Concept

When a `for` loop successfully executes without any break statement execution, then the `else` block attached with the `for` loop gets executed as well. But, if the `for` loop encounters a break during iteration, the `else` part won't execute.

```
for i in range(1,5):
    print('Welcome to Edureka',i)
    if(i>=5):
        break
else:
    print('The for loop was successfully executed')
```

```
Welcome to Edureka 1
Welcome to Edureka 2
Welcome to Edureka 3
Welcome to Edureka 4
The for loop was successfully executed
```

versus

```
for i in range(1,5):
    print('Welcome to Edureka',i)
    if(i>=4):
        break
else:
    print('The for loop was successfully executed')
```

```
Welcome to Edureka 1
Welcome to Edureka 2
Welcome to Edureka 3
Welcome to Edureka 4
```

Nested Loops

Nested loop, basically means a loop inside a loop. It can be a `for` loop inside a `while` loop and vice-versa. It can also be a `while` loop inside a `while` loop or `for` loop inside a `for` loop.

```
count=1
for i in range(10):
    print(str(i)*i)

    for j in range(0,i):
        count=count+1
```

`for` loop inside a `for` loop.

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

Loop Control Statements

Loop control statements are used to alter the execution flow from its normal flow.

Control Statement	Description
`break` statement	Is used to terminate the loop and the execution flow goes to the statement immediately following the loop.
`continue` statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
`pass` statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

edureka!

Loop Control Statements (contd.)

```
for i in range(10,50):
    print(i)
    if(i==30):
        break
```

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

```
for j in range(1,11):
    print(j)
    if(j==5):
        continue
```

```
1
2
3
4
5
6
7
8
9
10
```

```
for k in range(1,3):
    pass
print("Loop ends here")
```

```
Loop ends here
```

edureka!

Conditional Statements and Loops (Demonstration)

Note: Refer to Module 1: Demo 2 file on LMS for detailed steps.

edureka!

Structural Pattern Matching

edureka!

Structural Pattern Matching: Basics

- Structural pattern matching has been added in the form of a **`match`** statement and **`case`** statements of patterns with associated actions.
- Patterns consist of sequences, mappings, primitive data types as well as class instances.
- It enables programs to extract information from complex data types, branch on the structure of data, and apply specific actions based on different forms of data.

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

edureka!

Structural Pattern Matching: Simple Values

We'll write a function called `http_error()` that accepts a status code. We'll then use the status code as a subject of the match statement and write a couple of cases for different conditions:

```
def http_error(status):
    match status:
        case 200:
            return 'OK'
        case 400:
            return 'Bad request'
        case 401 | 403 | 404:
            return 'Not allowed'
        case _:
            return 'Something is wrong'
```

```
http_error(404)
'Not allowed'
http_error(200)
'OK'
http_error(12345)
'Something is wrong'
http_error('abc')
'Something is wrong'
```

Here we can see that the function doesn't crash even if you pass in a string value.

Structural Pattern Matching: Complex Patterns

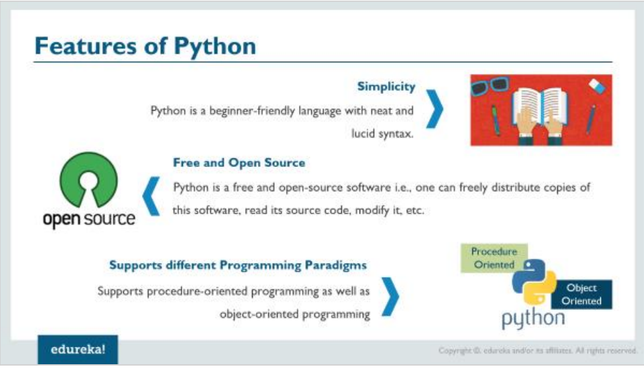
The code snippet below declares a function called `get_service_level()` that accepts user data as a dictionary. The goal is to return a different service level, based on the user **subscription type** (free, premium) and **type of the message** (info, error).

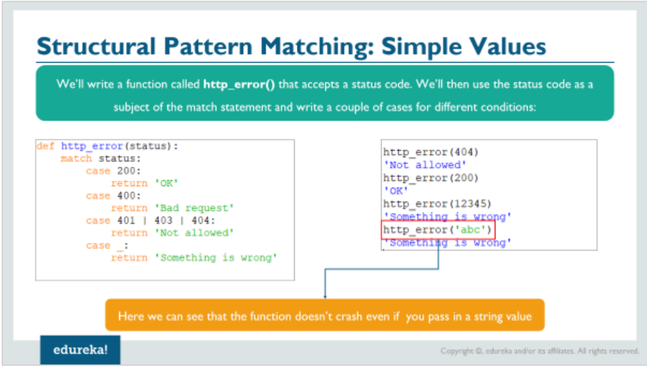
```
def get_service_level(user_data: dict):
    match user_data:
        case {'subscription': _, 'msg_type': 'info'}:
            print('Service level = 0')
        case {'subscription': 'free', 'msg_type': 'error'}:
            print('Service level = 1')
        case {'subscription': 'premium', 'msg_type': 'error'}:
            print('Service level = 2')
```

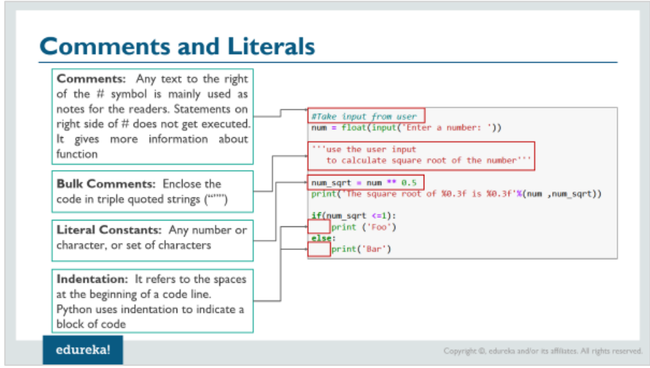
```
get_service_level({'subscription': 'free', 'msg_type': 'info'})
Service level = 0
get_service_level({'subscription': 'premium', 'msg_type': 'info'})
Service level = 0
get_service_level({'subscription': 'free', 'msg_type': 'error'})
Service level = 1
get_service_level({'subscription': 'premium', 'msg_type': 'error'})
Service level = 2
```

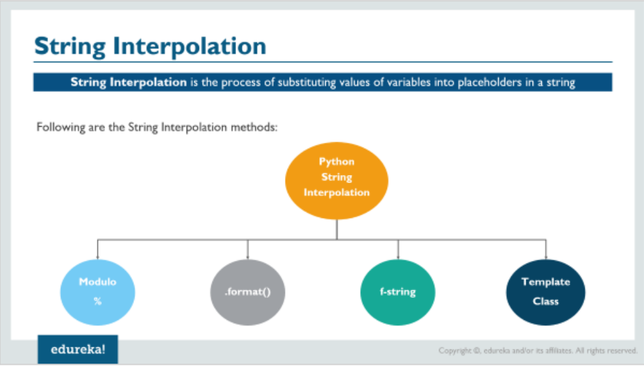
edureka!

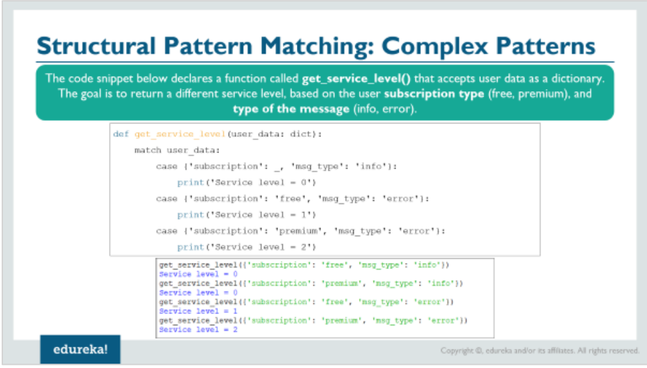
Summary

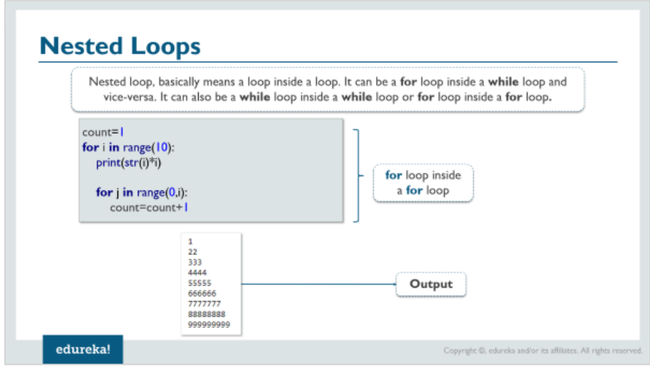












Questions

edureka!

edureka!

Thank You

For more information please visit our website
www.edureka.co