Introduction to Python

What is Python?

Python is one of the most popular programming languages. It was created by Guido van Rossum, and released in 1991. Python is a high-level, interpreted programming language known for its simplicity and readability, and is widely used in various domains, including web development, data analysis, artificial intelligence, and deep learning.

Python includes a vast array of libraries and frameworks (e.g., TensorFlow,- Keras, PyTorch) that simplify the implementation of deep learning models.

In this lecture, we will explore the fundamentals of Python programming.

What is Python?

Here is the first code example using Python:

```
print("Hello, World!")
```

```
Hello, World!
```

Python syntax can be executed by writing directly in the Command Line, or by creating a python file on the server, using the ***.py*** file extension, and running it in the Command Line.

```
>>> print("Hello, World!")
Hello, World!
```

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line. Although in other programming languages, the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

```
if 5 > 2:
  print("Five is greater than two!")
```

The following code will raise an error, because there is no indentation.

```
if 5 > 2:
print("Five is greater than two!")
```

We have to use the same number of spaces in the same block of code, otherwise there will be an error. For example, the following code will cause an error.

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

However, the following code will not raise any errors.

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

Python Comments

Comments can be used to make the code more readable, or to prevent execution when testing the code.
Comments start with a **#** sign, and Python will ignore the rest of the line.
Here are a few examples of comments in Python.

```
#This is a comment
print("Hello, World!")
```

```
print("Hello, World!") #This is a comment
```

```
#print("Hello, World!")
print("Hello, Friend!")
```

```
Hello, Friend!
```

Python Variables

Variables are containers for storing data values. A variable is created the moment we first assign a value to it.

```
x = 5
y = "Mary"
print(x)
print(y)
```

```
5
Mary
```

Variables are not declared with any particular type, and can change type after they have been set.

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
Sally
```

We can specify the data type of a variable. This is called **Casting**.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

We can get the data type of a variable using the ***type()*** function.

```
x = 5
y = "Mary"
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'str'>
```

Python Variables

String variables can be declared either by using single or double quotes.

```
x = "Mary"
print(x)
#double quotes are the same as single quotes:
x = 'Mary'
print(x)
```

```
Mary
Mary
```

There are a few rules for naming variables:

- Variable names must start with a letter or the underscore character.
- Variable names cannot start with a number.

- Variable names can only contain alpha-numeric characters and underscores.
- Variable names are case-sensitive (name, Name and NAME are three different variables).
- Variable names cannot be any of the Python keywords.

We can assign values to multiple variables in one line.

```
x, y = "Orange", "Banana"
print(x)
print(y)
```

```
Orange
Banana
```

Python Variables

We can assign the same value to multiple variables in one line.

```
x = y = "Orange"
print(x)
print(y)
```

```
Orange
Orange
```

To output variables, we can use the ***print()*** function.

```
x = "Orange"
print(x)
```

```
Orange
```

We can output multiple variables, separated by a comma. We can also use the ***+*** operator.

```
x = "Mary"
y = "likes"
z = "orange"
print(x, y, z)
```

```
Mary likes orange
```

```
x = "Mary "
y = "likes "
z = "orange"
print(x + y + z)
```

```
Mary likes orange
```

For numbers, the ***+*** operator functions as a mathematical operator.

```
x = 5
y = 10
print(x + y)
```

```
15
```

Python Variables

If we try to combine a string and a number with the + operator, we will get an error. The best way to output multiple variables with different types, is to separate them by commas.

```
x = "Mary is"
y = 5
print(x, y)
```

```
Mary is 5
```

Variables can store data of different types, and different types are used for different purposes.
Here are the built-in data types in Python:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Python Numbers

Variables of numeric types are created when we assign a value to them. Here are a few points about numbers in Pythons:

- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
- Float, or a Floating Point number is a number, positive or negative, containing one or more decimals.
- Floats can also include an ***e***, indicating the power of 10.
- Complex numbers are written with a ***j***, indicating the imaginary part.

```
#int
x1 = 1
x2 = -2564821
print(x1)
print(x2)
```

```
1
-2564821
```

```
#float
y1 = 2.8
y2 = -3.4
y3 = 12E4
y4 = -87.7e5
print(y1)
print(y2)
print(y3)
print(y4)
```

```
2.8
-3.4
120000.0
-8770000.0
```

```
#complex
z1 = 1j
z2 = 5 -2j
print(z1)
print(z2)
```

```
1j
(5-2j)
```

Python Strings

Variables of string type are created when we assign a value to them. Here are a few points about strings in Pythons:

- We can use quotes inside a string, as long as they don't match the quotes surrounding the string.
- The ***split()*** method returns a list where the text between the specified separator becomes the list items.
- To insert a new line in a string, we use the ***\n*** escape character. There are multiple escape characters that can be used for different purposes.

```
print("It's alright")
print("He is called 'Johnny'")
print('He is called "Johnny"')
```

```
It's alright
He is called 'Johnny'
He is called "Johnny"
```

```
a = "Hello, World!"
b = a.split(",")
print(b)
```

```
['Hello', ' World!']
```

```
txt = "She is Mary.\nMary is 5 years old."
print(txt)
```

```
She is Mary.
Mary is 5 years old.
```

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace

Python Booleans

Booleans represent one of two values, True or False. Here are a few points about strings in Pythons:

- We can evaluate any expression in Python, and get one of two answers, True or False.
- When we compare two values, the expression is evaluated and Python returns the Boolean answer.
- The ***bool()*** function allows us to evaluate any value, and it returns either True or False.
- Almost any value is evaluated to True if it has some sort of content.

- Any string is True, except empty-strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

```
print(bool("Hello"))
print(bool(15))
```

```
True
True
```

Python Operators

Operators are used to perform operations on variables and values. Here are the Python operators:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Python Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Python Operators

Here is a tables that demonstrates operator precedence in Python, starting with the highest precedence at the top:

Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Python if/elif/else

An if statement can be written using the ***if*** keyword. Similarly, an else statement is written using the ***else*** keyword. Here are a few points about if and else statements:

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.
- The ***elif*** keyword in Python means "if the previous conditions were not true, then try this condition".
- The ***else*** keyword catches anything which is not caught by the preceding conditions.

```
a = 33
b = 200

if b > a:
  print("b is greater than a")
```

```
b is greater than a
```

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

Python While Loops

Python has two primitive loop commands, ***while*** loops and ***for*** loops.
Using the while loop we can execute a set of statements as long as a condition is true. Here are a few points about while loops:

- With the ***break*** statement, we can stop a loop, even if the while condition is true.
- With the ***continue*** statement, we can stop the current iteration, and continue with the next.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1
```

```
1
2
3
```

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

# Note that number 3 is missing in the result
```

```
1
2
4
5
6
```

Python For Loops

A for loop is used for iterating over a sequence. We can have nested loops. Here are a few points about for loops:

- With the ***break*** statement, we can stop the loop before it has looped through all the items.
- With the ***continue*** statement, we can stop the current iteration of the loop, and continue with the next.
- To loop through a set of code for a specified number of times, we can use the ***range()*** function.
- Loops cannot be empty, but if we have an empty for loop, using the ***pass*** statement will prevent an error.

```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

```
for x in "banana":
  if x == 'n':
    break
  print(x)
```

```
b
a
```

```
for x in "banana":
  if x == 'n':
    continue
  print(x)
```

```
b
a
a
a
```

```
#range(start_number(default = 0), end_number(exclusive), step(default = 1))

for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

```
for x in [0, 1, 2]:
  pass
```