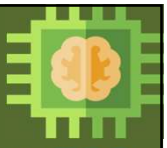
Elective Course

Course Code: CS4103

Autumn 2025-26



Lecture #02

Artificial Intelligence for Data Science

Week-1: INTRODUCTION TO ARTIFICIAL INTELLIGENCE (AI)

Python Primer for AI (Part-I)

Course Instructor:

Dr. Monidipa Das

Assistant Professor

Department of Computational and Data Sciences

Indian Institute of Science Education and Research Kolkata, India 741246

Topics

- Identifiers
- Keywords
- Lists, Strings, Tuples
- Dictionary
- Conditional Statements
- Looping Statements
- Functions
- Modules and Packages

..based on some materials taken from CS1101

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module, etc.

Valid	Invalid
Computer	1computer
_computer	computer@
__computer__	yield
comp_12	lambda

- Python is a case sensitive programming language; **Computer** and **computer** are two different identifiers in Python

Python Keywords

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

```
In [35]: import keyword
    ...: print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Lines and Indentation

- Blocks of code are denoted by line indentation
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

```
if True:
            print("Write answer")
            print("True")
else:
            print("Write answer")
            print("False")
```


Quotation and Comments

Multi-Line Statements:

```
total =  item_one + \
          item_two + \
          item_three

days = ['Monday', 'Tuesday',
'Friday']
```

Quotation in Python:

```
   word = 'word'
   sentence = "This is a sentence."
   paragraph = """This is a paragraph. It is made up
of multiple lines and sentences."""
```

Comments in Python:

A hash sign (#) that is not inside a string literal begins a comment

Python Variable Types

- Variables are nothing but reserved memory locations to store values.
- Python variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable
- Python data types:
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary

```
counter = 100 # An integer assignment
miles = 99.5  # A floating point
name = "Nandini" # A string
```


Python Numbers

- Number data types store numeric values. They are immutable data types

```
a=100
print(a,id(a))
a=a+11
print(a,id(a))
```

```
100 140734700167048
111 140734700167400
```

- Python supports different numerical types:
  - int
  - float
  - complex

```
a=100
b=99.9
c=2+3j
print(type(a),type(b),type(c))
<class 'int'> <class 'float'> <class 'complex'>
```

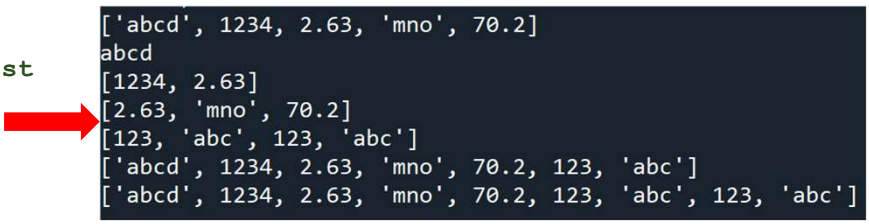
Python Lists

- A list contains items separated by commas and enclosed within square brackets ([])
- The items belonging to a list can be of different data type.

```
lst = [ 'abcd', 1234 , 2.63, 'mno', 70.2 ]
tinylist = [123, 'abc']

print(lst)
print(lst[0])
#Slice operator on list
print(lst[1:3])
print(lst[2:])

#Algebra with lists
print(tinylist * 2)
print(lst + tinylist)
print(lst + 2*tinylist)
```



```
['abcd', 1234, 2.63, 'mno', 70.2]
abcd
[1234, 2.63]
[2.63, 'mno', 70.2]
[123, 'abc', 123, 'abc']
['abcd', 1234, 2.63, 'mno', 70.2, 123, 'abc']
['abcd', 1234, 2.63, 'mno', 70.2, 123, 'abc', 123, 'abc']
```

- A list is mutable

```
print(tinylist[1])
tinylist[1]='ABC'
print(tinylist[1])
```

```
abc
ABC
```

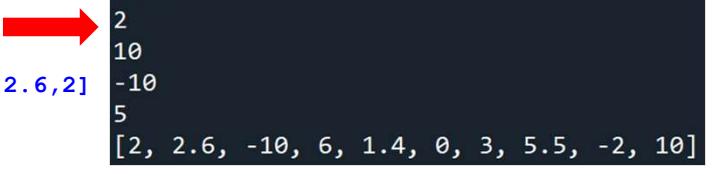

Python Lists [contd.]


```
#List Length, Sum and Average
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
print(len(numlist))
print(sum(numlist))
print(sum(numlist)/len(numlist))

#Location of an element in the list
print(numlist.index(5.5))

#Finding Maximum and Minimum
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
print(max(numlist))
print(min(numlist))
print(numlist.index(1.4))

#Reversing the order of elements in the list
numlist.reverse()
print(numlist)
```



```
10
18.5
1.85
2
10
-10
5
[2, 2.6, -10, 6, 1.4, 0, 3, 5.5, -2, 10]
```

Python Lists [contd.]


```
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
#Sorting a list
numlist.sort()
print(numlist)
numlist.sort(reverse=True)
print(numlist)
```

```
[-10, -2, 0, 1.4, 2, 2.6, 3, 5.5, 6, 10]
[10, 6, 5.5, 3, 2.6, 2, 1.4, 0, -2, -10]
```

```
#Sorting a list without changing the list by using the sorted command
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
l1=sorted(numlist)
print(numlist)
print(l1)
l2=sorted(numlist, reverse=True)
print(numlist)
print(l2)
```

```
[10, -2, 5.5, 3, 0, 1.4, 6, -10, 2.6, 2]
[-10, -2, 0, 1.4, 2, 2.6, 3, 5.5, 6, 10]
[10, -2, 5.5, 3, 0, 1.4, 6, -10, 2.6, 2]
[10, 6, 5.5, 3, 2.6, 2, 1.4, 0, -2, -10]
```


Python Lists [contd.]


```
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]

#Adding item to the end of the List
numlist.append(5)
print(numlist)

#Adding item to a list at a given index position using insert
numlist.insert(2,8)
print(numlist)
```

```
[10, -2, 5.5, 3, 0, 1.4, 6, -10, 2.6, 2, 5]
[10, -2, 8, 5.5, 3, 0, 1.4, 6, -10, 2.6, 2, 5]
```

Python Lists [contd.]


```
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
#Removing item from a list from the right side
numlist.pop()
print(numlist)
numlist.pop()
print(numlist)

#Removing a specific item from a list using remove
numlist.remove(3)
print(numlist)

#Removing items from a list at specific positions using del
numlist = [10,-2,5.5,3,0,1.4,6,-10,2.6,2]
del numlist[2:6]
print(numlist)
```

```
[10, -2, 5.5, 3, 0, 1.4, 6, -10, 2.6]
[10, -2, 5.5, 3, 0, 1.4, 6, -10]
[10, -2, 5.5, 0, 1.4, 6, -10]
[10, -2, 6, -10, 2.6, 2]
```


Python List Comprehension

- A concise and efficient way to create new lists based on existing iterables (lists, tuples, strings, ranges)

```
#Creating a new list from an existing iterable
L1=[1,2,3,4,5]
print(L1)
L2=[i**2 for i in L1]
print(L2)
L3=[i**2 for i in L1 if i<=4]
print (L3)
L4=[i for i in "Intelligence"]
print (L4)
#Creating a list of pairs of numbers from two lists
L5=[1,2,3]
L6=['a','b','c']
L7=[(i,j) for i in L5 for j in L6]
print(L7)
```

```
[1, 2, 3, 4, 5]
[1, 4, 9, 16, 25]
[1, 4, 9, 16]
['I', 'n', 't', 'e', 'l', 'l', 'i', 'g', 'e', 'n', 'c', 'e']
[(1, 'a'), (1, 'b'), (1, 'c'), (2, 'a'), (2, 'b'), (2, 'c'), (3, 'a'), (3, 'b'), (3, 'c')]
```

Python Strings

- Strings in Python are identified as a contiguous set of characters in between quotation marks.

```
str='Welcome'
print(str)
print(str[0])
print(str[2:6])
print(str[2:])
print(str[-2:])
print(str[-1:-4:-1])
print(str[::-1])

#Arithmetic operations with a string
print(str * 2)
#String concatenation
print(str + " All!")
```

```
Welcome
W
lcom
lcome
me
emo
emocleW
WelcomeWelcome
Welcome All!
```


Python Strings [contd.]


```
#Splitting a string by blank spaces
text = "The term Artificial Intelligence was coined in 1956"
textaslist = text.split()
print (textaslist)
print (textaslist[2])
text = 'The term, Artificial Intelligence, was coined in 1956'
#Splits at ','
print (text.split(','))
text = 'The term Artificial Intelligence was coined in: 1956'
#Splitting at ':'
print (text.split(':'))
#Rejoining the split string
joined=" ".join(textaslist)
print(joined)
joined="-".join(textaslist)
print(joined)
```

```
['The', 'term', 'Artificial', 'Intelligence', 'was', 'coined', 'in', '1956']
Artificial
['The term', ' Artificial Intelligence', ' was coined in 1956']
['The term Artificial Intelligence was coined in', ' 1956']
The term Artificial Intelligence was coined in 1956
The-term-Artificial-Intelligence-was-coined-in-1956
```

Python Strings [contd.]


```
numstr='2025'
numstr2='10'
#Converting numeric string into a
#number
print(eval(numstr))
print(eval(numstr+numstr2))
print(eval(numstr)+eval(numstr2))

#You can convert a string to a
#number using eval command only if
#the string is numeric
s="abc"
print(eval(s))
```

```
2025
202510
2035
Traceback (most recent call last):

  Cell In[65], line 9
    print(eval(s))

  File <string>:1

NameError: name 'abc' is not defined
```


Python Tuples

- A tuple is an **immutable** list that cannot be changed ➔ elements cannot be inserted or deleted from a tuple.

```
t1=(0,1,2,3,4)
t2=(9,8,7,6,5)
dept=('CDS','DBS','DCS','DES','DMS','DPS')
print(dept[0])
print(t1+t2)
print(len(t1+t2))

#print(t2.sort()) #Attribute error
print(sorted(t2))
print(sorted(t1+t2,reverse=True))
l=[5,3,0,-2,4,-1,6]
print(tuple(l)) #list to tuple
print(list(dept)) #tuple to list
```

```
CDS
(0, 1, 2, 3, 4, 9, 8, 7, 6, 5)
10
[5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
(5, 3, 0, -2, 4, -1, 6)
['CDS', 'DBS', 'DCS', 'DES', 'DMS', 'DPS']
```

Python Dictionary

- A dictionary is an unordered set of keys along with values associated with the specified keys
- A dictionary is **mutable**

```
D={'BFS':2,'DFS':6,'UCS':3}
print (D)
print(D.keys())
print(D.values())
print(D.items())

if 'DFS' in D:
    print(D['DFS'])
else:
    print ('DFS is absent')

D['A*']=2
print(D)
print (sorted(D.keys()))
del D['DFS']
print(D)
```

```
{'BFS': 2, 'DFS': 6, 'UCS': 3}
dict_keys(['BFS', 'DFS', 'UCS'])
dict_values([2, 6, 3])
dict_items([('BFS', 2), ('DFS', 6), ('UCS', 3)])
6
{'BFS': 2, 'DFS': 6, 'UCS': 3, 'A*': 2}
['A*', 'BFS', 'DFS', 'UCS']
{'BFS': 2, 'UCS': 3, 'A*': 2}
```


Python *if...elif...else* Statement

- The syntax of the if statement is:

```
if expression:
    statement(s)
```

```
if expression:
    statement(s)
else:
    statement(s)
```

```
v1 = 0
if v1:
        print("1 - Got a true expression value")
        print(v1)
v2 = 77
if v2:
        print("2 - Got a true expression value")
        print(v2)
print("Good bye!")
```

```
2 - Got a true expression value
77
Good bye!
```

Nested *if...elif...else* Construct

Example:

```
m = 100
if m < 200:
   print("Expression value is less than 200")
   if m == 150:
      print("Which is 150")
   elif m == 100:
      print("Which is 100")
   elif m == 50:
      print("Which is 50")
elif m < 500:
   print("Expression value is less than 500")
else:
   print("Expression value is larger")
print("Good bye!")
```

```
Expression value is less than 200
Which is 100
Good bye!
```


Python *while* Loop Statements

- The **while** loop continues until the expression becomes false.
  The syntax of the while loop is:

```
while expression:
      statement(s)
```

Example:

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
print("End of loop")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
End of loop
```

Python *for* Loop Statements

- The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- The syntax of the loop look is:

```
for iterating_var in sequence:
    statements(s)
```

Example:

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

for letter in 'Python':
   print('Current Letter :', letter)
print("End of loop")

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:
   print('Current fruit :', fruit)
print("End of loop")
```



```
1 1
2 4
3 9
4 16
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
End of loop
Current fruit : banana
Current fruit : apple
Current fruit : mango
End of loop
```


break and *continue* Statements

- The **break** statement in Python terminates the current loop and resumes execution at the next statement
- The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

Example:

```
for letter in 'Python':
   if letter == 'h':
          break
   print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

Example:

```
for letter in 'Python':
   if letter == 'h':
      continue
   print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

The *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.
- The **pass** statement is a *null* operation;

Example:

```
for letter in 'Python':
   if letter == 'h':
      pass
   else:
       print('Current Letter :',letter)
print("Exited!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Exited!
```


Python Function

- Define functions to avoid code repetition and for simplicity

```
2
1729
(5, 3)
```

- A function can be called multiple times with different arguments
- Indentation is essential

```
#A function with single argument
def last_digit(x):
    if x>=10:
        x = x%10
    return x
print(last_digit(102))
#A function with two arguments
def square_add(x,y):
    z = x**3 + y**3
    return z
print(square_add(9,10))
#A function returning multiple values
def f(x,y):
    x = x + y
    y = x - y
    x = x - y
    return x,y
print(f(3,5))
```

Python Function

- A function can also be defined without the def statement by using the 'lambda function'

```
f=lambda x,y: x**3 + y**3
print(f(1,12))

print((lambda x: x+25)(2025))
```

Python: Modules and Packages

- **Module:** a simple Python file that contains collections of functions, classes, and global variables and with having a .py extension file.

```
d={'A':65,'B':66,'C':67}

class Abc:
    def __init__(self,year):
        self.yr=year

def func_myModule(name):
    print("Module : "+ name)
```

mymodule.py

```
import mymodule
mymodule.func_myModule("Test")
a=mymodule.Abc(2025)
print(a.yr)
print(mymodule.d)
```

```
Module : Test
2025
{'A': 65, 'B': 66, 'C': 67}
```

- **Package:** A directory containing modules and an __init__.py file.

A Few Python Libraries for AI

29

30

Questions?