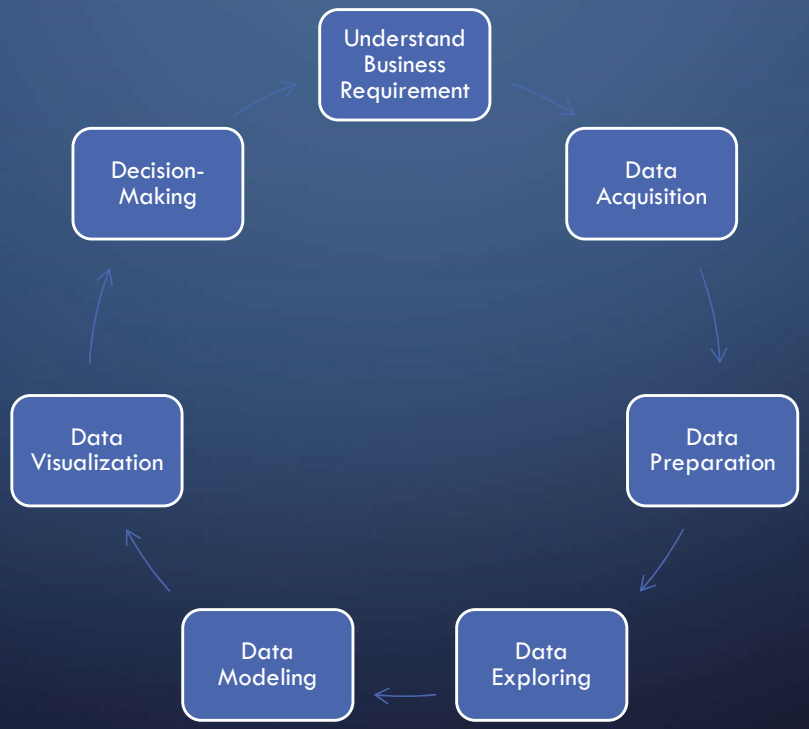
INTRODUCTION TO DATA SCIENCE WITH PYTHON

CHAPTER 1

WHAT IS DATA SCIENCE?

- Data science is the field that comprises everything related to cleaning, preparing, and analyzing unstructured, semistructured, and structured data.
- This field of science uses a combination of statistics, mathematics, programming, problem-solving, and data capture to extract insights and information from data.

THE STAGES OF DATA SCIENCE



WHERE DATA SCIENCE BEING IMPLEMENTED?

- Internet search engine – to deliver the best results for search queries in less time.
- Recommendation systems that use a user's experience to generate recommendation.
- Digital advertisement.
- Education systems.
- Healthcare systems.
- And many more!

WHY PYTHON?

- Python is a dynamic and general-purpose programming language that is used in various fields.
- Python applications – GUI and database programming, client and server-side applications, and application testing, even automation.
- It was initially developed in the early 1990s by Guido van Rossum and is now controlled by the not-for-profit Python Software Foundation, sponsored by Microsoft, Google and others.

BASIC FEATURES OF PYTHON

- Easy to learn and use
- Expressive
- Interpreted
- Cross-platform
- Free and open source
- Object-oriented
- Extensible
- Large standard library
- GUI programming support
- Integrated

PYTHON ENVIRONMENT AND EDITORS

- Install python
- Install IDE
  - Anaconda
  - VS Code
  - Jupyter
  - Google Colab
  - Spyder
  - PyCharm
  - Etc

BASIC SYNTAX

- Identifier – a name used to identify a variable, function, class, module, or other object in the script.
- Identifier starts with a letter (A-Z or a-z) or an underscore (_) followed by more letters, numbers or underscores.
- Python is a case-sensitive language.

PYTHON CONVENTION

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two leading underscores, the identifier is a language-defined special name.

PYTHON RESERVE KEYWORDS

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

LINES AND INDENTATION

- Line indentation is important in Python because Python does not depend on braces to indicate blocks of code for class and function definitions or flow control.
- Therefore, a code segment block is denoted by line indentation, which is rigidly enforced.

MULTILINE STATEMENTS

- Statements in Python typically end with a new line.
- But a programmer can use the line continuation character (\) to denote that the line should continue.

QUOTATION MARKS IN PYTHON

- Python accepts single ('), double ("), and triple ''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- However, triple quotes are used to span the string across multiple lines.

MULTIPLE STATEMENTS ON A SINGLE LINE

- Python allows the use of \n to split line into multiple lines.
- In addition, the semicolon (;) allows multiple statements on a single line if neither statement starts a new code block.

```
In [9]: TV=15; name="Nour"; print (name); print ("Welcome
to\nDubai Festival 2018")

Nour
Welcome to
Dubai Festival 2018
```

READ DATA FROM USERS

```
In [10]:name = input("Enter your name ")
         age = int (input("Enter your age "))
         print ("\nName =", name); print ("\nAge =", age)
```

```
Enter your name Nour
Enter your age 12

Name = Nour

Age = 12
```

DECLARING VARIABLES AND ASSIGNING VALUES

- Python has five standard data types that are used to define the operations possible on them and the storage method for each of them.
  - Number
  - String
  - List
  - Tuple
  - Dictionary

MULTIPLE ASSIGNS

```
In [13]:age= mark = code =25
        print (age)
        print (mark)
        print (code)
```

```
In [14]:age, mark, code=10,75,"CIS2403"
        print (age)
        print (mark)
        print (code)
```

VARIABLE NAMES AND KEYWORDS

- A variable is an identifier that allocates specific memory space and assigns a value that could change during the program runtime.
- Variable names should refer to the usage of the variable, so if you want to create a variable for student age, then you can name it as age or student_age.

STATEMENTS AND EXPRESSIONS

- A statement is any unit of code that can be executed by a Python interpreter to get a specific result or perform a specific task.
- A program contains a sequence of statements, each of which has a specific purpose during program execution.
- The expression is a combination of values, variables, and operators that are evaluated by the interpreter to do a specific task

BASIC OPERATORS IN PYTHON

- Arithmetic operators
- Relational operators
- Assign operators
- Logical operators
- Membership operators
- Identify operators
- Bitwise operators

ARITHMETIC OPERATORS

Operators	Description	Example	Output
//	Performs floor division (gives the integer value after division)	`print (13//5)`	2
+	Performs addition	`print (13+5)`	18
-	Performs subtraction	`print (13-5)`	8
*	Performs multiplication	`print (2*5)`	10
/	Performs division	`print (13/5)`	2.6
%	Returns the remainder after division (modulus)	`print (13%5)`	3
**	Returns an exponent (raises to a power)	`print (2**3)`	8

RELATIONAL OPERATORS

Operators	Description	Example	Output
<	Less than	`print (13<5)`	False
>	Greater than	`print (13>5)`	True
<=	Less than or equal to	`print (13<=5)`	False
>=	Greater than or equal to	`print (2>=5)`	False
==	Equal to	`print (13==5)`	False
!=	Not equal to	`print (13! =5)`	True

ASSIGN OPERATORS

Operators	Description	Example	Output
=	Assigns	x=10 print (x)	10
/=	Divides and assigns	x=10; x/=2 print (x)	5.0
+=	Adds and assigns	x=10; x+=7 print (x)	17
-=	Subtracts and assigns	x=10; x-=6 print (x)	4

ASSIGN OPERATORS

Operators	Description	Example	Output
*=	Multiplies and assigns	x=10; x*=5 print (x)	50
%=	Modulus and assigns	x=13; x%=5 print (x)	3
**=	Exponent and assigns	x=10; x**=3 print(x)	1000
//=	Floor division and assigns	x=10; x//=2 print(x)	5

LOGICAL OPERATORS

Operators	Description	Example	Output
and	Logical AND (when both conditions are true, the output will be true)	`x=10>5 and 4>20` `print (x)`	False
or	Logical OR (if any one condition is true, the output will be true)	`x=10>5 or 4>20` `print (x)`	True
not	Logical NOT (complements the condition; i.e., reverses it)	`x=not (10<4)` `print (x)`	True

PYTHON COMMENTS

```
In [18]: # Python single line comment
In [19]: ''' This
         Is
         Multi-line comment'''
```

FORMATTING STRINGS

- The Python special operator % helps to create formatted output.
- This operator takes two operands, which are a formatted string and a value.

```
In [20]: print ("pi=%s"%"3.14159")

pi=3.14159
```

CONVERSION TYPES

Syntax	Description
%c	Converts to a single character
%d, %i	Converts to a signed decimal integer or long integer
%u	Converts to an unsigned decimal integer
%e, %E	Converts to a floating point in exponential notation
%f	Converts to a floating point in fixed-decimal notation
%g	Converts to the value shorter of %f and %e
%G	Converts to the value shorter of %f and %E
%o	Converts to an unsigned integer in octal
%r	Converts to a string generated with `repr()`
%s	Converts to a string using the `str()` function
%x, %X	Converts to an unsigned integer in hexadecimal

THE REPLACEMENT FIELD, {}

- You can use the replacement field, {}, as a name (or index).
- If an index is provided, it is the index of the list of arguments provided in the field.

```
In [24]:x = "price is"
         print ("{1} {0} {2}".format(x, "The", 1920.345))

The price is 1920.345
```

THE DATE AND TIME MODULE

```
In [42]:import time localtime = time.asctime(time.
localtime(time.time()))
print ("Formatted time :", localtime)
print(time.localtime())
print (time.time())

Formatted time : Fri Aug 17 19:12:07 2018

time.struct_time(tm_year=2018, tm_mon=8, tm_mday=17,
tm_hour=19, tm_min=12, tm_sec=7, tm_wday=4, tm_yday=229,
tm_isdst=0)

1534533127.8304486
```

TIME MODULE METHODS

Methods	Description
`time()`	Returns time in seconds since January 1, 1970.
`asctime(time)`	Returns a 24-character string, e.g., Sat Jun 16 21:27:18 2018.
`sleep(time)`	Used to stop time for the given interval of time.
`strptime (String,format)`	Returns a tuple with nine time attributes. It receives a string of date and a format. `time.struct_time(tm_year=2018, tm_mon=6, tm_mday=16, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=177, tm_isdst=-1)`

(continued)

TIME MODULE METHODS

Methods	Description
`gtime()/ gtime(sec)`	Returns `struct_time`, which contains nine time attributes.
`mktime()`	Returns the seconds in floating point since the epoch.
`strftime (format)/ strftime (format,time)`	Returns the time in a particular format. If the time is not given, the current time in seconds is fetched.

PYTHON CALENDAR MODULE

Methods	Description
`prcal(year)`	Prints the whole calendar of the year.
`firstweekday()`	Returns the first weekday. It is by default 0, which specifies Monday.
`isleap(year)`	Returns a Boolean value, i.e., true or false. Returns true in the case the given year is a leap year; otherwise, false.
`monthcalendar(year,month)`	Returns the given month with each week as one list.
`leapdays(year1,year2)`	Returns the number of leap days from `year1` to `year2`.
`prmonth(year,month)`	Prints the given month of the given year.

SELECTION STATEMENTS

Form	if statement	if-else Statement	Nested if Statement

Structure	`if(condition):`
	`statements`
		`if(condition):`
		`statements`
		`else:`
		`statements`
			`if (condition):`
			`statements`
			`elif (condition):`
			`statements`
			`else:`
			`statements`

ITERATION STATEMENTS

1	`for` **loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	**Nested loops** You can use one or more loop inside any another `while`, `for`, or `do..while` loop.
3	`while` **loop** Repeats a statement or group of statements while a given condition is true. It tests the condition *before* executing the loop body.
4	`do {....} while ()` Repeats a statement or group of statements while a given condition is true. It tests the condition *after* executing the loop body.

LOOP CONTROL STATEMENTS

1	**Break statement** Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	**Continue statement** Causes the loop to skip the remainder of its body and immediately retests its condition prior to reiterating.
3	**Pass statement** The pass statement is used when a statement is required syntactically but you do not want any command or code to execute.

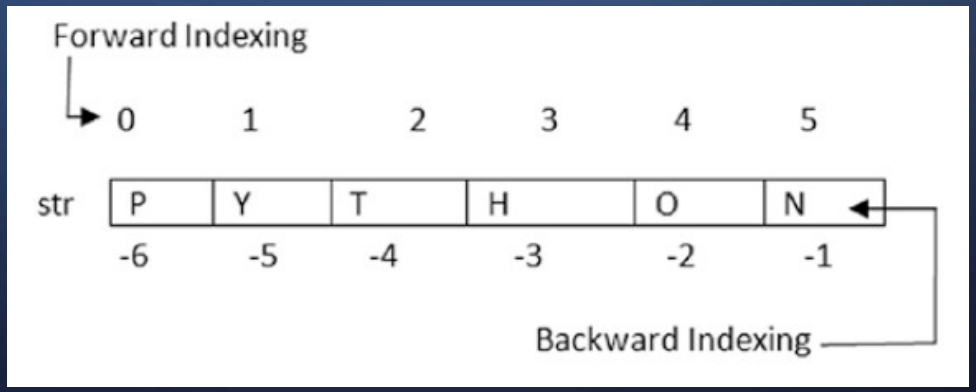
TRY AND EXCEPT

- try and except are used to handle unexpected values where you would like to validate entered values to avoid error occurrence.

```
In [14]: # Try and Except
astr='Al Fayoum'
      errosms=''
try:
            istr=int(astr) # error
except:
            istr=-1
            errosms="\nIncorrect entry"
print ("First Try:", istr , errosms)
```

STRING PROCESSING

- A string is a sequence of characters that can be accessed by an expression in brackets called an index.
- Python considers strings by enclosing text in single as well as double quotes.



STRING SPECIAL OPERATORS

Operator	Description	Outputs
+	Concatenation: adds values on either side of the operator	`a + b` will give `HelloPython`.
*	Repetition: creates new strings, concatenating multiple copies of the same string	`a*2` will give `-HelloHello`.
[]	Slice: gives the character from the given index	`a[1]` will give `e`.
[:]	Range slice: gives the characters from the given range	`a[1:4]` will give `ell`.
in	Membership: returns true if a character exists in the given string	`H in a` will give true.
not in	Membership: returns true if a character does not exist in the given string	`M not in a` will give true.

STRING FORMAT SYMBOLS

Format Symbol	Conversion
%c	Character
%s	String conversion via `str()` prior to formatting
%i	Signed decimal integer
%d	Signed decimal integer
%u	Unsigned decimal integer

(continued)

Format Symbol	Conversion
%o	Octal integer
%x	Hexadecimal integer (lowercase letters)
%X	Hexadecimal integer (uppercase letters)
%e	Exponential notation (with lowercase e)
%E	Exponential notation (with uppercase E)
%f	Floating-point real number
%g	The shorter of %f and %e
%G	The shorter of %f and %E

STRING SLICING AND CONCATENATION

- String slicing refers to a segment of a string that is extracted using an index or using search methods.
- In addition, the len() method is a built-in function that returns the number of characters in a string.
- Concatenation enables you to join more than one string together to form another string.

STRING CONVERSIONS AND FORMATTING SYMBOLS

```
In [14]:#Convert string to int
       str3 = '123'
       str3= int (str3)+1
       print (str3)

124

In [15]:#Read and convert data
       name=input('Enter your name: ')
       age=input('Enter your age: ')
       age= int(age) + 1
       print ("Name: %s"% name ,"\t Age:%d"% age)

Enter your name: Omar
Enter your age: 41

Name: Omar     Age:42
```

PYTHON STRING FUNCTIONS AND METHODS

Method/Function	Description
`capitalize()`	Capitalizes the first character of the string.
`count(string, begin,end)`	Counts a number of times a substring occurs in a string between the beginning and end indices.
`endswith(suffix, begin=0,end=n)`	Returns a Boolean value if the string terminates with a given suffix between the beginning and end.

(continued)

Method/Function	Description
find(substring, beginIndex, endIndex)	Returns the index value of the string where the substring is found between the begin index and the end index.
index(subsring, beginIndex, endIndex)	Throws an exception if the string is not found and works same as the find() method.
isalnum()	Returns true if the characters in the string are alphanumeric (i.e., letters or numbers) and there is at least one character. Otherwise, returns false.
isalpha()	Returns true when all the characters are letters and there is at least one character; otherwise, false.
isdigit()	Returns true if all the characters are digits and there is at least one character; otherwise, false.
islower()	Returns true if the characters of a string are in lowercase; otherwise, false.
isupper()	Returns false if the characters of a string are in uppercase; otherwise, false.
isspace()	Returns true if the characters of a string are white space; otherwise, false.
len(string)	Returns the length of a string.
lower()	Converts all the characters of a string to lowercase.
upper()	Converts all the characters of a string to uppercase.
startswith(str, begin=0,end=n)	Returns a Boolean value if the string starts with the given str between the beginning and end.

(continued)

Method/Function	Description
swapcase()	Inverts the case of all characters in a string.
lstrip()	Removes all leading white space of a string and can also be used to remove a particular character from leading white spaces.
rstrip()	Removes all trailing white space of a string and can also be used to remove a particular character from trailing white spaces.

THE IN OPERATOR

- The word in is a Boolean operator that takes two strings and returns true if the first appears as a substring in the second.

```
In [43]:var1 =' Higher Colleges of Technology '
       var2='College'
       var3='g'
       print ( var2 in var1)
       print ( var2 not in var1)

True
False
```

TABULAR DATA AND DATA FORMATS

- Data is available in different forms. It can be unstructured data, semistructured data, or structured data.
- Python provides different structures to maintain data and to manipulate it such as variables, lists, dictionaries, tuples, series, panels, and data frames.

PANDAS DATAFRAME

- A Pandas data frame can be created using the following constructor:

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

- A Pandas data frame can be created using various input forms such as the following:

List	Dictionary	Series
Numpy ndarrays	Another dataframe	

PYTHON PANDAS DATA SCIENCE LIBRARY

- Provides a mechanism to load data objects from different formats.
- Creates efficient data frame objects with default and customized indexing.
- Reshapes and pivots date sets.
- Provides efficient mechanisms to handle missing data.
- Merges, groups by, aggregates, and transforms data.
- Manipulates large data sets by implementing various functionalities such as slicing, indexing, subsetting, deletion, and insertion.
- Provides efficient time series functionality.

A PANDAS SERIES

- A series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, Python objects, etc.).

A PANDAS DATA FRAME

- A data frame is a two-dimensional data structure.
- In other words, data is aligned in a tabular fashion in rows and columns.

A PANDAS PANELS

- A panel is a 3D container of data that can be created from different data structures such as from a dictionary of data frames.

PYTHON LAMBDAS AND THE NUMPY LIBRARY

- The lambda operator is a way to create small anonymous functions, in other words, functions without names.
- These functions are throwaway functions; they are just needed where they have been created.
- Lambda functions are used in combination with the functions filter(), map(), and reduce().

ANONYMOUS FUNCTIONS

- Anonymous functions refer to functions that aren't named and are created by using the keyword lambda.
- A lambda is created without using the def keyword; it takes any number of arguments and returns an evaluated expression.

THE MAP() FUNCTION

- The map() function is used to apply a specific function on a sequence of data.
- The map() function has two arguments.

```
r = map(func, seq)
```

THE FILTER() FUNCTION

- The filter() function is an elegant way to filter out all elements of a list for which the applied function returns true.

```
filter(func, list1)
```

THE REDUCE () FUNCTION

- The reduce() function continually applies the function func to a sequence seq and returns a single value.
- The reduce() function is used to find the max value in a sequence of integers.

PYTHON NUMPY PACKAGE

- Numpy is a Python package that stands for “numerical Python.”
- It is a library consisting of multidimensional array objects and a collection of routines for processing arrays.
- The Numpy library is used to apply the following operations:
  - Operations related to linear algebra and random number generation
  - Mathematical and logical operations on arrays
  - Fourier transforms and routines for shape manipulation

DATA CLEANING AND MANIPULATION TECHNIQUES

- Keeping accurate data is highly important for any data scientist.
- Developing an accurate model and getting accurate predictions from the applied model depend on the missing values treatment.
- Therefore, handling missing data is important to make models more accurate and valid.

DATA CLEANING AND MANIPULATION TECHNIQUES

- Numerous techniques and approaches are used to handle missing data such as the following:
  - Fill NA forward
  - Fill NA backward
  - Drop missing values
  - Replace missing (or) generic values
  - Replace NaN with a scalar value

RUNNING BASIC INFERENTIAL ANALYSES

- Linear regression
- Finding correlation
- Measuring central tendency
- Measuring variance
- Normal distribution
- Binomial distribution
- Poisson distribution
- Bernoulli distribution
- Calculating p-value
- Implementing a Chi-square test

SUMMARY

- The data science main concepts and life cycle.
- The importance of Python programming and its main libraries used for data science processing.
- Different Python data structure use in data science applications.
- How to apply basic Python programming techniques.
- Initial implementation of abstract series and data frames as the main Python data structure.
- Data cleaning and its manipulation techniques.
- Running basic inferential statistical analyses.