University of Sulaimani
College of Science
Computer Department
4th Stage

Data Science Management

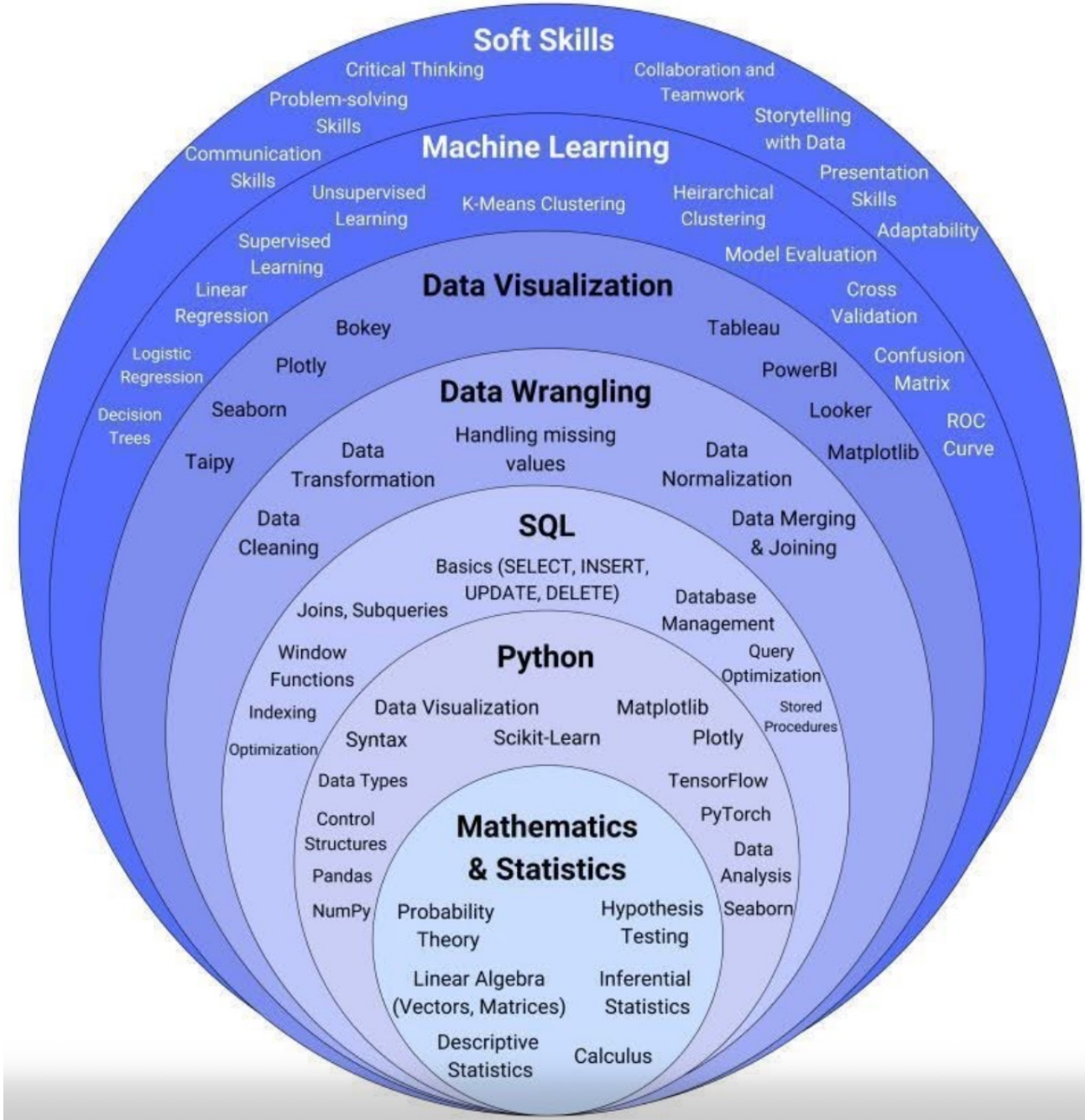
Introduction to Data Science

Class 1

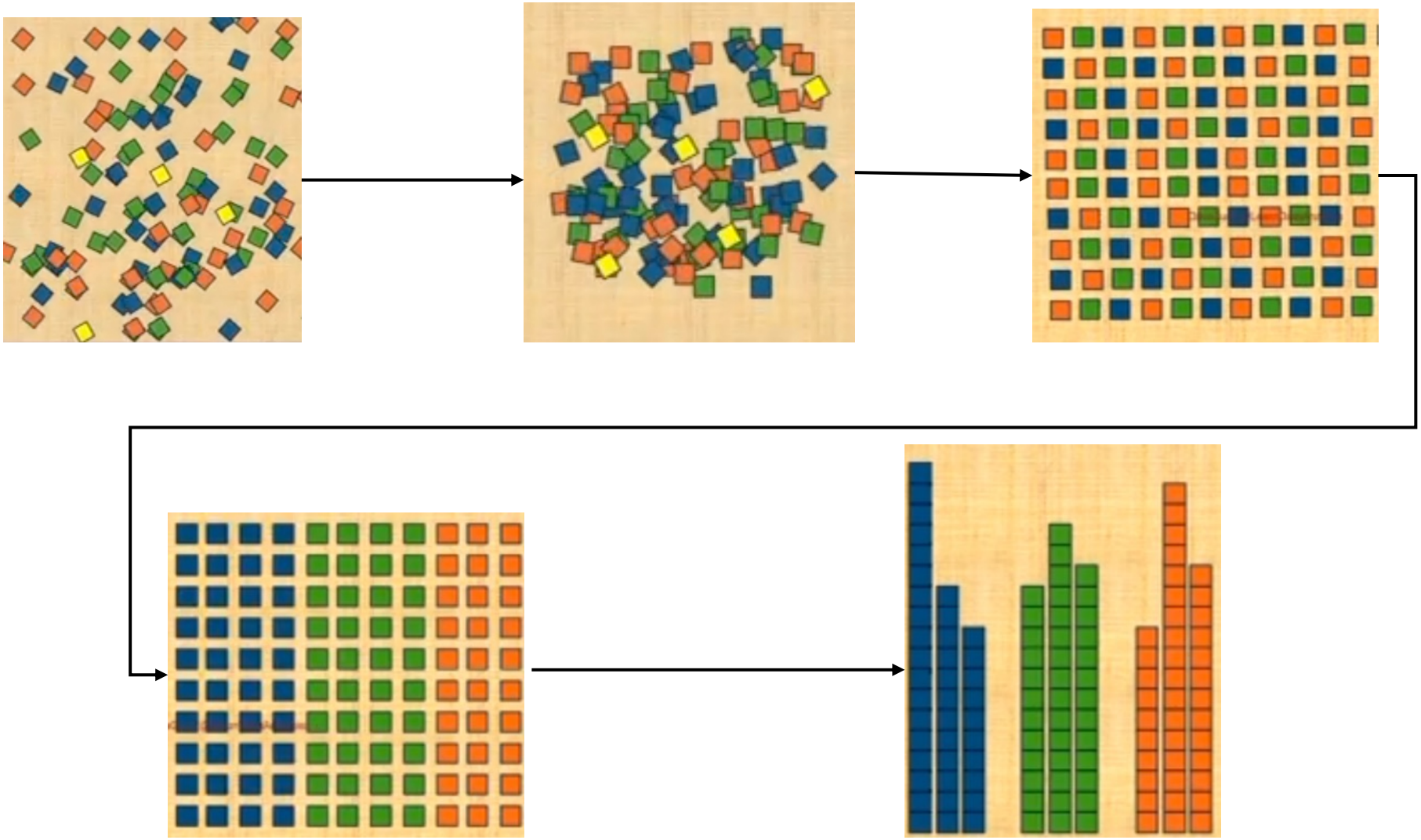
Theoretical and practical lectures

Assist. Prof. Dr. Miran Taha Abdullah
2025-2026

Data Scientist Roadmap



What are your thoughts on Data Science



The agenda for this lecture includes:

What is Data Science?

Why Data Science?

What Does a Data Scientist Do?

Why Become a Data Scientist

Skills Required to Become a Data Scientist

Data Scientist vs Data Analyst

Average Salary of a Data Scientist

Data Scientist Roadmap

Introduction to Data Science

- Data science is the field of study that uses:

scientific methods, Processes, algorithms, and systems

to **extract knowledge** and **insights** from **structured** and **unstructured** data.

The scientific method is the process of *objectively* establishing facts by testing and experimenting. It's a systematic way to ensure that findings are based on evidence rather than assumptions or bias

Processes : a series of actions or steps taken in order to achieve a particular end.

Algorithm : step-by-step set of instructions used to solve a specific problem or perform a task

A **system** is an organized collection of components or elements that interact and function together to achieve a common objective

- **Data** is a collection of information.
- One purpose of Data Science is to structure data, making it interpretable and easy to work with.
- Data can be categorized into two groups:

- **Unstructured Data**
- Unstructured data is not organized. We must organize the data for analysis purposes.

Structured Data

Structured data is organized and easier to work with.

Unstructured data

Structured data

Introduction to Data Science

- Data Science combines various disciplines such as **statistics**, **machine learning**, **data analysis**, and **visualization** to **uncover hidden patterns**, **trends**, and **correlations in data**.

- Data science plays a crucial role in **decision-making**, **forecasting**, and **problem-solving** across industries, driving innovation and enabling organizations to make data-driven decisions

What is Data Science?

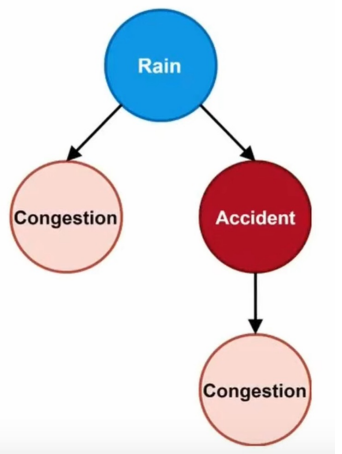
- Data Science is about **data gathering**, **analysis** and **decision-making**.
- Data Science is **about finding patterns in data**, through **analysis**, and make future **predictions**.

By applying Data Science, companies can achieve:

- Better decisions (should we choose A or B)
- Predictive analysis (what will happen next?)
- Pattern discoveries (find pattern, or maybe hidden information in the data)

Better decisions

Predictive analysis



Pattern discoveries

Where is Data Science Needed?

- Data Science is used in many industries in the world today,

e.g. banking, consultancy, healthcare, and manufacturing.

<u>Examples of where Data Science is needed:</u>

- For route planning: To discover the best routes to ship.
- To foresee delays for flight/ship/train etc. (through predictive analysis).
- To create promotional offers.
- To find the best suited time to deliver goods.
- To forecast the next years revenue for a company.
- To analyze health benefit of training.
- To predict who will win elections.

Data science's lifecycle

1- Capture: This stage involves gathering raw structured and unstructured data.

i.e. Data Acquisition, Data Entry, Signal Reception, Data Extraction.

2- Maintain: This stage covers taking the raw data and putting it in a form that can be used.

i.e. Data Warehousing, Data Cleansing, Data Staging, Data Processing, Data Architecture.

3- Process: Data scientists take the prepared data and examine its patterns, ranges, and biases to determine how useful it will be in predictive analysis.

i.e. Data Mining, Clustering/Classification, Data Modelling, Data Summarization.

Data science's lifecycle (cont.)

4- Analyse: This stage involves performing the various analyses on the data.

i.e. Exploratory/Confirmatory, Predictive Analysis, Regression, Text Mining, Qualitative Analysis.

5- Communicate: In this final step, analysts prepare the analyses in easily readable forms such as charts, graphs, and reports.

i.e. Data Reporting, Data Visualization, Business Intelligence, Decision Making.

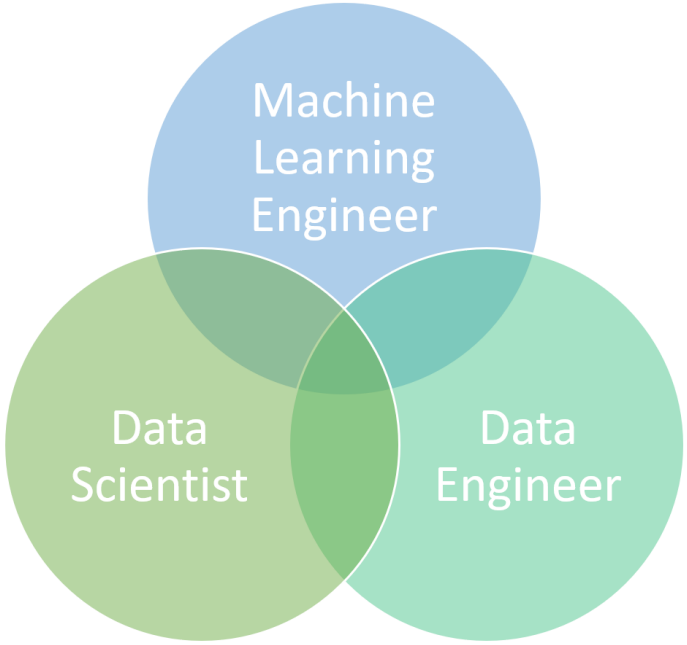
How Does a Data Scientist Operate?

1. **Ask the right questions** - To understand the business problem.
2. **Explore and collect data** - From database, web logs, customer feedback, etc.
3. **Extract the data** - Transform the data to a standardized format.
4. **Clean the data** - Remove erroneous values from the data.
5. **Find and replace missing values** - Check for missing values and replace them with a suitable value (e.g. an average value).
6. **Normalize data** - Scale the values in a practical range (e.g. 140 cm is smaller than 1,8 m. However, the number 140 is larger than 1,8. - so scaling is important).
7. **Analyze data, find patterns and make future predictions**.
8. **Represent the result** - Present the result with useful insights in a way the "company" can understand.

Types of Data Science

If you learn data science, then you get the opportunity to find the various exciting roles in this domain. The main roles are given below:

1. Data Scientist
2. Data Analyst
3. Machine learning expert
4. Data engineer
5. Data Architect
6. Data Administrator
7. Business Analyst
8. Business Intelligence Manager



Types of Data Science (cont.)

1. Data Analyst:

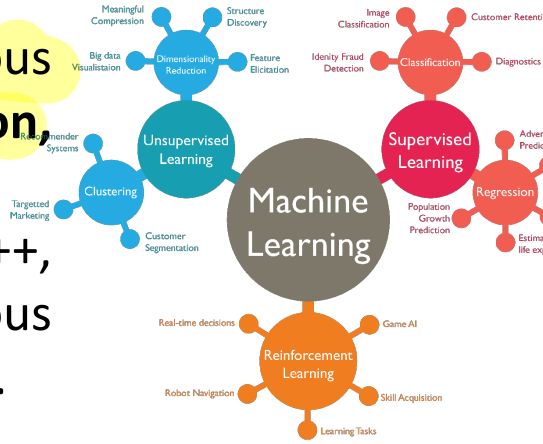
Data analyst is an individual, who performs mining of huge amount of data, models the data, looks for patterns, relationship, trends, and so on. They come up with visualization and reporting for analyzing the data for decision making and problem-solving process.

Skill required: For becoming a data analyst, you must get a good background in **mathematics, business intelligence, data mining**, and basic knowledge of **statistics**. You should also be familiar with some computer languages and tools such as **MATLAB, Python, SQL, Hive, Pig, Excel, SAS, R, JS, Spark**, etc.

2. Machine Learning Expert:

The machine learning expert is the one who works with various machine learning algorithms used in data science such as **regression, clustering, classification, decision tree, random forest**, etc.

Skill Required: Computer programming languages such as Python, C++, R, Java, and Hadoop. You should also have an understanding of various algorithms, problem-solving analytical skill, probability, and statistics.



3. Data Engineer:

A data engineer works with massive amount of data and responsible for building and maintaining the data architecture of a data science project. Data engineer also works for the creation of data set processes used in modelling, mining, acquisition, and verification.

Skill required: Data engineer must have depth knowledge of **SQL, MongoDB, Cassandra, HBase, Apache Spark, Hive, MapReduce**, with language knowledge of **Python, C/C++, Java, Perl**, etc.

Data Engineers
They own the data realms.
They build systems.
They make the data usable.

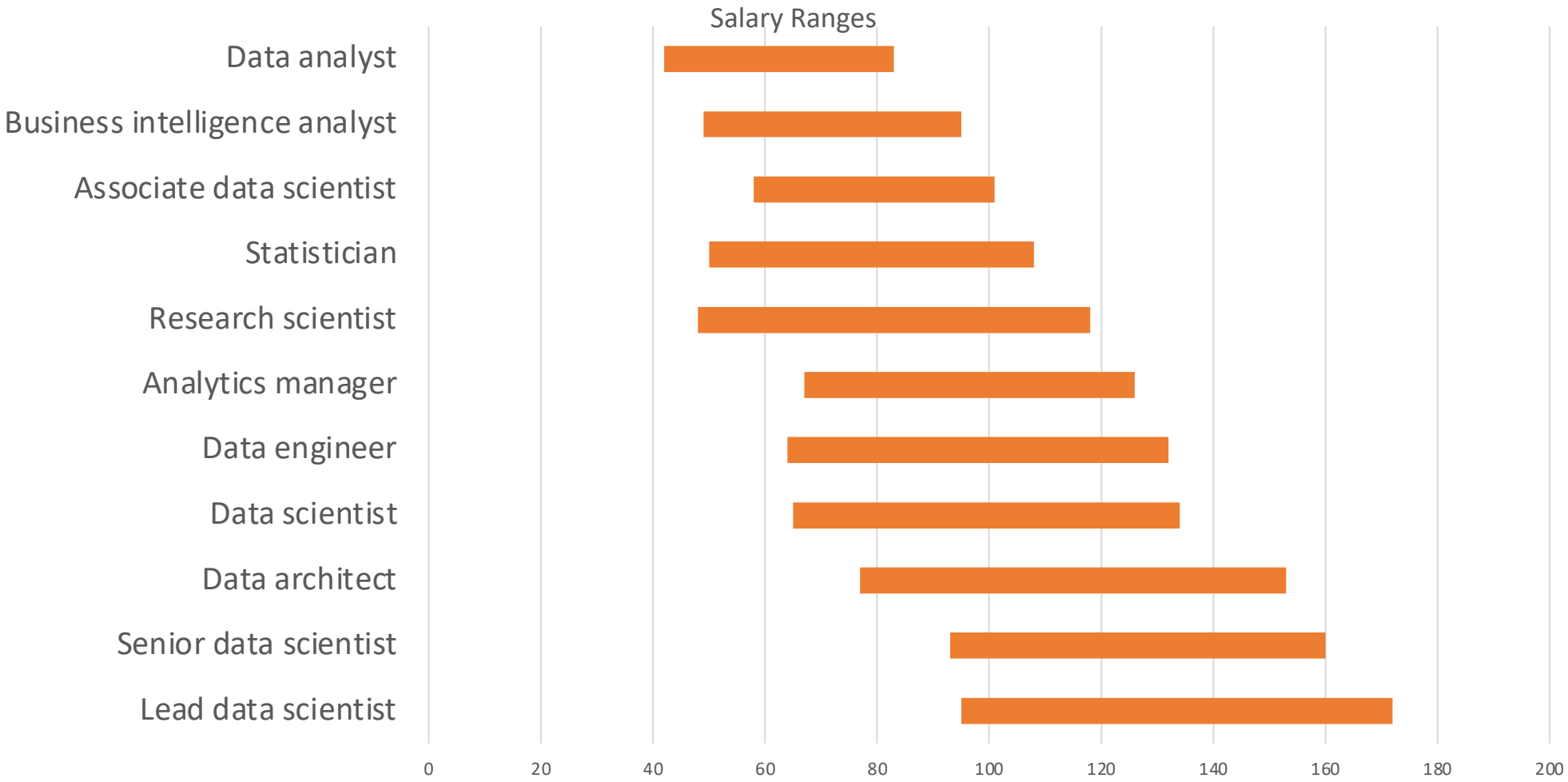
4. Data Scientist:

A data scientist is a professional who works with an enormous amount of data to come up with compelling business insights through the deployment of various tools, techniques, methodologies, algorithms, etc.

Skill required: one should have technical language skills such as **R, SAS, SQL, Python, Hive, Pig, Apache spark, MATLAB**. Data scientists must have an understanding of Statistics, Mathematics, visualization, and communication skills.

Data Scientists
They make insights out of data and build models that allow you to automate business processes or decisions.

Relation between Salary and skills



Prerequisite for Data Science (How Does a Data Scientist Work?)

Non-Technical Prerequisite

Curiosity: To learn data science, one must have curiosities. When you have curiosity and ask various questions, then you can understand the business problem easily.

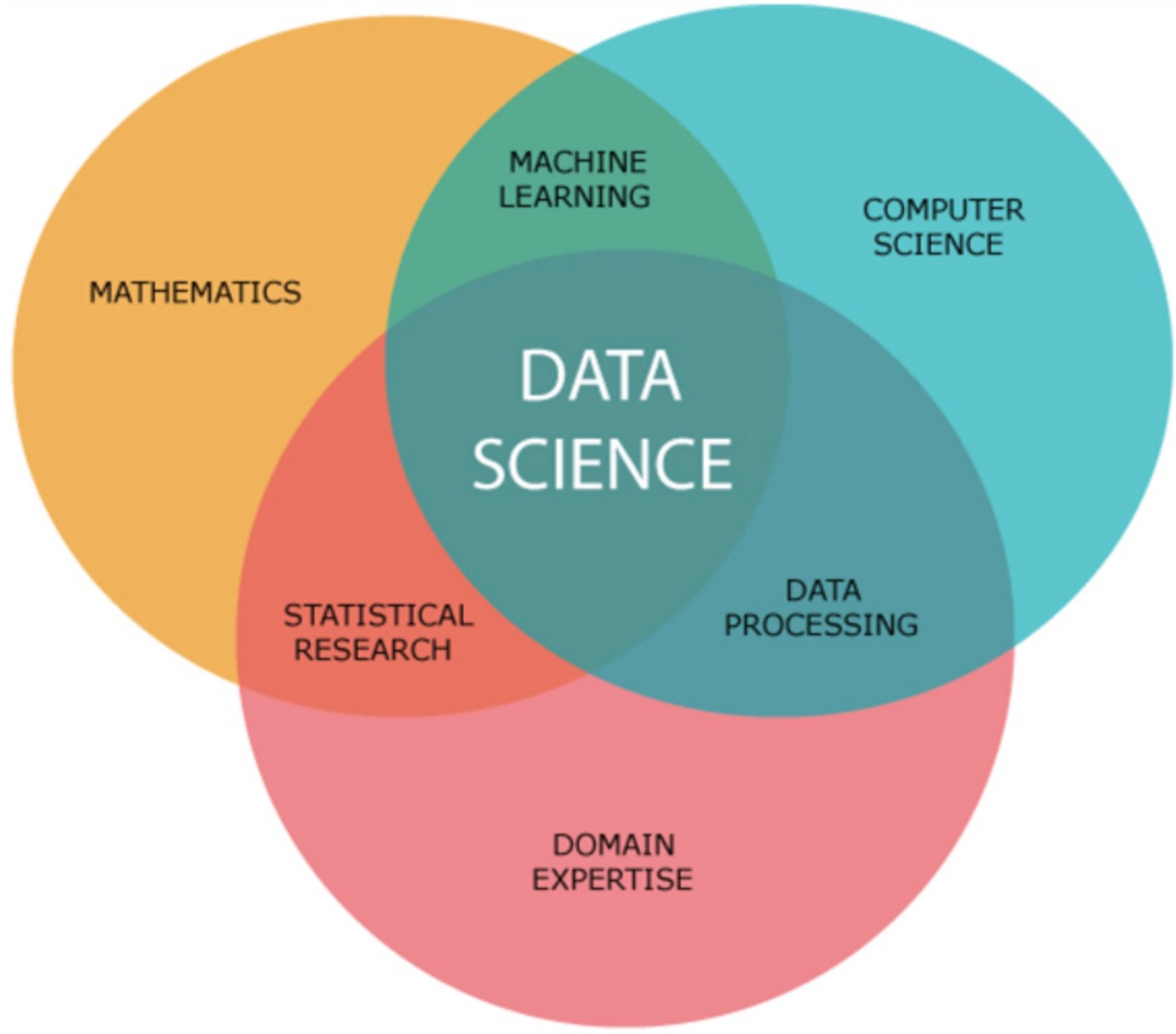
Critical Thinking: It is also required for a data scientist so that you can find multiple new ways to solve the problem with efficiency.

Communication skills: Communication skills are most important for a data scientist because after solving a business problem, you need to communicate it with the team

Technical Prerequisite

- **Machine learning:** To understand data science, one needs to understand the concept of machine learning. Data science uses machine learning algorithms to solve various problems.
- **Mathematical modelling:** Mathematical modelling is required to make fast mathematical calculations and predictions from the available data.
- **Statistics:** Basic understanding of statistics is required, such as mean, median, or standard deviation. It is needed to extract knowledge and obtain better results from the data.
- **Computer programming:** For data science, knowledge of at least one programming language is required. R, Python, Spark are some required computer programming languages for data science.
- **Databases:** The depth understanding of Databases such as SQL, is essential for data science to get the data and to work with data.

Main components of Data Science



Data Scientist

Roadmap

Mathematics

- Linear Algebra
- Analytics Geometry
- Matrix
- Vector Calculus
- Optimization
- Regression
- Dimensionality Reduction
- Density Estimation
- Classification

Probability

- Discrete Distribution
  - *Binomial*
  - *Bernouli*
  - *Geometric etc*
- Continuos Distribution
  - *Uniform*
  - *Exponential*
  - *Gamma*
- Normal Distribution
- Introduction to Probability
- 1D Random Variable
- Function of One Random Variable
- Joint Probability Distribution

Statistics

- Introduction to Statistics
- Data Description
- Random Samples
- Sampling Distribution
- Parameter Estimation
- Hypotheses Testing
- ANOVA
- Reliability Engineering
- Stochastic Process
- Computer Simulation
- Design of Experiments
- Simple Linear Regression
- Correlation
- Multiple Regression
- Nonparametric Statistics
  - *Sign Test*
  - *The Wilcoxon Signed-Rank Test*
  - *The Wilcoxon Rank Sum Test*
  - *The Kruskal-Walls Test*
- Statistical Quality Control
- Basic of Graphs

Programming

Python

- Python Basics
  - *List*
  - *Set*
  - *Tuples*
  - *Dictionary*
  - *Function, etc.*
- NumPy
- Pandas
- Matplotlib/Seaborn, etc.

R

- R Basic
  - *Vector*
  - *List*
  - *Data Frame*
  - *Matrix*
  - *Array, etc*
- dplyr
- ggplot2
- Tidyr
- Shiny, etc.

DataBase

- SQL
- MongoDB

Other

- Data Structure
  - *Array, etc*
- Web Scraping
- Linux
- Git

Machine Learning

Introduction

- How Model Works
- Basic Data Exploration
- First ML Model
- Model Validation
- Underfitting & Overfitting
- Random Forests
- scikit-learn

Intermediate

- Handling Missing Values
- Handling Categorical Variables
- Pipelines
- Cross-Validation
- XGBoost
- Data Leakage

Deep Learning

- Artificial Neural Network
- Convolutional Neural Network
- Recurrent Neural Network
- Keras
- PyTorch
- TensorFlow
- A Single Neuron
- Deep Neural Network
- Stochastic Gradient Descent
- Overfitting and Underfitting
- Dropout Batch Normalization
- Binary Classification

Feature Engineering

- Baseline Model
- Categorical Encodings
- Feature Generation
- Feature Selection

Natural language Processing

- Text Classification
- Word Vectors

Data Visualization Tools

- Excel VBA
- Bi (Business Intelligence)
  - *Tableau*
  - *Power BI*
  - *Qlik View*
  - *Qlik Sense*

Deployment

- Microsoft Azure
- Heroku
- Google Cloud Platform
- Flask
- Django

Other Points

- Domain Knowledge
- Communication Skill
- Reinforcement Learning
- Case Studies
  - *Data Science at Netflix*
  - *Data Science at Flipkart*
  - *Project on Credit Card Fraud Detection*
  - *Project on Movie Recommendation , etc.*

Keep Practicing

Python
For Everything

Python	+	Pandas	=	Data manipulation
Python	+	TensorFlow	=	Deep learning
Python	+	Matplotlib	=	Data visualization
Python	+	Seaborn	=	Advanced charts
Python	+	BeautifulSoup	=	Web scraping
Python	+	Selenium	=	Browser automation
Python	+	FastAPI	=	High-performance APIs
Python	+	SQLAlchemy	=	Database access
Python	+	Flask	=	Lightweight web apps
Python	+	Django	=	Scalable platforms
Python	+	OpenCV	=	Game development

Why python?

- High-level interpreted language
- Very clear syntax => easy-to-read
- Open source
- Platform independent
- Large number of libraries
- Binding to all standard GUI kits (TkInter, Qt, ..)
- Powerful automated testing tools
- Easily integrated with C/C++, and Fortran
- Actively used and extended by scientists

Python Language

- Clear syntax -> easy to understand
- Indentation for blocks
- Use of namespace, exception handling
- Automatic doc.: doc strings & pydoc
- Dynamic typing
- Supports both procedural & OO approach
- Easy code testing
- Garbage collecting
- Functional programming, iterators,…

Who is using Python

SPACE TELESCOPE SCIENCE INSTITUTE

Data processing and calibration for instruments on the Hubble Space Telescope.

HOLLYWOOD

Digital animation and special effects:

- Industrial Light and Magic
- Imageworks
- Tippett Studios
- Disney
- Dreamworks

PETROLEUM INDUSTRY

Geophysics and exploration tools:

- ConocoPhillips, Shell

GOOGLE

One of top three languages used at Google along with C++ and Java. Guido works there.

PAINT SHOP PRO 9

Scripting Engine for JASC PaintShop Pro photo-editing software

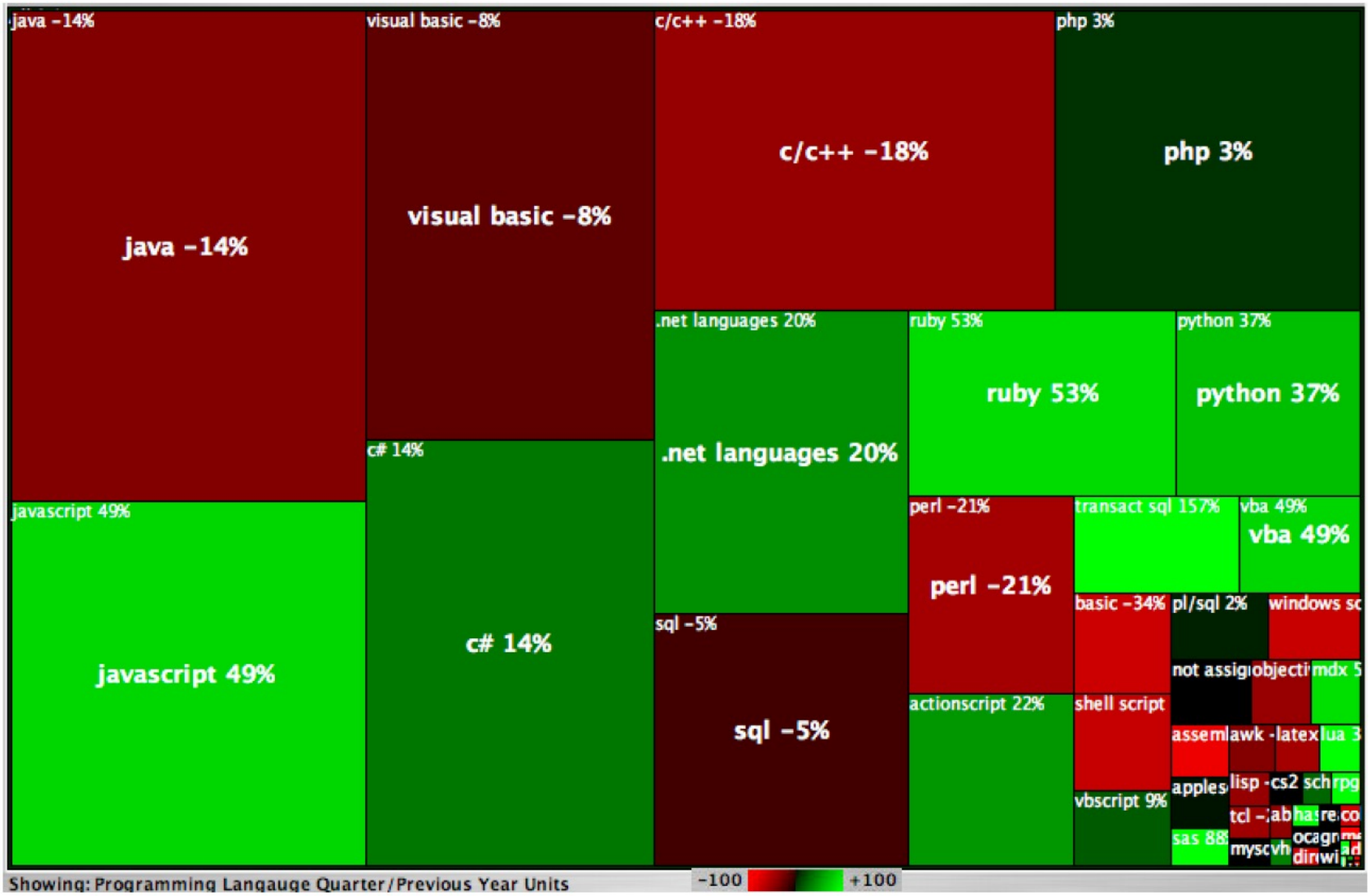
REDHAT

Anaconda, the Redhat Linux installer program, is written in Python.

PROCTER & GAMBLE

Fluid dynamics simulation tools.

Programming Language Market



Advanced interactive environments:

- **IPython**, an advanced **Python console** http://ipython.org/
- **Jupyter**, **notebooks** in the browser http://jupyter.org/
- **Anaconda** offers the easiest way to perform Python/R data science and machine learning, https://www.anaconda.com/
- Spyder: integrates an IPython console, a debugger, a profiler… https://www.spyder-ide.org
- PyCharm: integrates an IPython console, notebooks, a debugger… (freely available, but commercial)
- Visual Studio Code: integrates a Python console, notebooks, a debugger, …
- Atom

Core Numeric libraries

- **Numpy**: numerical computing with powerful **numerical arrays** objects, and routines to manipulate them. http://www.numpy.org/

Pandas - This library is used for structured data operations, like import CSV files, create dataframes, and data preparation

- **Scipy**: high-level numerical routines. Optimization, regression, interpolation, etc http://www.scipy.org/

- **Matplotlib**: 2-D visualization, “publication-ready” plots http://matplotlib.org/

Domain-specific packages,

- **Mayavi** for 3-D visualization
- **pandas, statsmodels, seaborn** for statistics
- **sympy** for symbolic computing
- **scikit-image** for image processing
- **scikit-learn** for machine learning

Students should correctly install those libraries on their devices.

- NumPy
- SciPy
- Pandas
- Matplotlib
- Keras
- SciKit-Learn
- PyTorch
- Scrapy
- BeautifulSoup
- TensorFlow

Class outline

1. **Basic Mathematical Operations**
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. Working with Fractions
5. Complex Numbers
6. Getting User Input
7. Handling Exceptions and Invalid Input

1.Basic Mathematical Operations

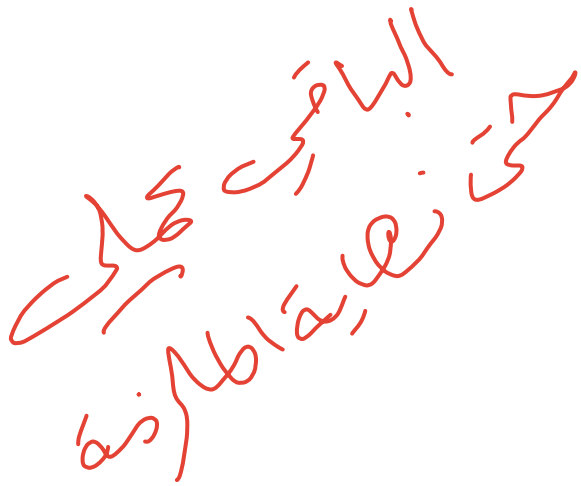
Python language supports following type of operators.

Arithmetic Operators , Comparison Operators , Logical (or Relational) Operators , Assignment Operators , Conditional (or ternary) Operators

Python Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

There are 7 arithmetic operators in Python :
Addition
Subtraction
Multiplication
Division
Modulus
Exponentiation
Floor division



1. Addition Operator : In Python, **+** is the addition operator. It is used to add 2 values.
Example :

```
val1 = 2
val2 = 3

# using the addition operator
res = val1 + val2
print(res)
```

Output : 5

2. Subtraction Operator : In Python, **–** is the subtraction operator. It is used to subtract the second value from the first value.
Example :

```
val1 = 2
val2 = 3

# using the subtraction operator
res = val1 - val2
print(res)
```

Output : -1

3. Multiplication Operator : In Python, ***** is the multiplication operator. It is used to find the product of 2 values.

Example :

```
val1 = 2
val2 = 3

# using the multiplication operator
res = val1 * val2
print(res)
```

Output : 6

4. Division Operator : In Python, **/** is the division operator. It is used to find the quotient when first operand is divided by the second.

Example :

```
val1 = 3
val2 = 2
# using the division operator
res = val1 / val2
print(res)
```

Output : 1.5

5. Modulus Operator : In Python, **%** is the modulus operator. It is used to find the remainder when first operand is divided by the second.

Example :

```
val1 = 3
val2 = 2

# using the modulus operator
res = val1 % val2
print(res)
```

Output : 1

6. Exponentiation Operator : In Python, ** is the exponentiation operator. It is used to raise the first operand to power of second.

Example :

```
val1 = 2
val2 = 3

# using the exponentiation operator
res = val1 ** val2
print(res)
```

Output : 8

7. Floor division : In Python, **//** is used to conduct the floor division. It is used to find the floor of the quotient when first operand is divided by the second.

Example :

```
val1 = 3
val2 = 2
# using the floor division
res = val1 // val2
print(res)
```

Output : 1

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y
**	Power : Returns first raised to power second	x ** y

What is the difference between Float Division and Floor Division

- / is normal division it will give floating value → 24/12 = 2.0 not 2
  # Floating Point Number

- // is floor division it will give integer value → 24 // 12 = 2 not 2.0
  # Whole Number

- >>> print 5.0 / 2 → 2.5
- >>> print 5.0 // 2 → 2.0

How the Modulus is operated in python

Dividend Quotient

10 / 3 = 3 r 1

Divisor Remainder

in programming the **modulo** operation finds the remainder of division of one number by another

in python the statement is written as

a % n

where a and n are positive integers

>>> 10 % 3 → 1

1 1 1 1 1 1 1 1 1 1 = 10

>>> 11 % 3 → 2

1 1 1 1 1 1 1 1 1 1 1 = 11

2 % 10 = 2 *If your n > a → your result is a*

3 % 0 = Undefined

```
def euclidean_division(x, y):
    quotient = x // y
    remainder = x % y
    print(f"{quotient} remainder {remainder}")

euclidean_division(10, 3)  # 3 remainder 1
euclidean_division(11, 3)  # 3 remainder 2
```

```
# Examples of Arithmetic Operator
a = 9
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b

# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
p = a ** b
```



```
# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(p)
```



```
13
 5
36
2.25
2
1
6561
```

Comparison Operators

Comparison of Relational operators compares the values.
It either returns **True** or **False** according to the **condition**.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y

```
# Examples of Relational Operators
a = 13
b = 33

# a > b is False
print(a > b)

# a < b is True
print(a < b)

# a == b is False
print(a == b)

# a != b is True
print(a != b)

# a >= b is False
print(a >= b)

# a <= b is True
print(a <= b)
```

Output

```
False
True
False
True
False
True
```

Logical Operators

Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to **combine conditional statements**.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```


Output

```
False
True
False
```

Bitwise Operators

Bitwise operators act on **bits and perform the bit-by-bit** operations. These are used to operate on **binary numbers.**

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

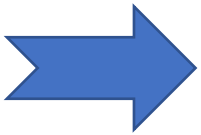
In Python how to convert a number to Binary

- Do you want to write a program to do it ?
- Answer: it's simple method, use the bin() function to convert from a decimal value to its corresponding binary value.

Use bin() Function to Convert Int to Binary in Python
Use format Function to Convert Int to Binary in Python
Use the str.format() Method to Convert Int to Binary in Python

- >>> bin(1)→ 0b**1**
- >>>bin(2)→ 0b**10**
- >>>bin(3)→ 0b**11**
- >>>bin(4)→ 0b**100**

format(***no.***, 'b')



```
>>> format(1,'b')
'1'
>>> format(2,'b')
'10'
>>> format(3,'b')
'11'
>>> format(4,'b')
'100'
```

Format types convertion

Formatting Type	Role
=	Places the sign to the leftmost position
b	Converts the value into equivalent binary
o	Converts value to octal format
x	Converts value to Hex format
d	Converts the given value to decimal
E	Scientific format, with an E in Uppercase
X	Converts value to Hex format in upper case

```
# Examples of Bitwise operators
a = 10
b = 4

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
print(~a)

# print bitwise XOR operation
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```


Output

```
0
14
-11
14
2
40
```

```
>>> format(10,'b')
'1010'
>>> format(4,'b')
'100'
```

Assignment Operators

Assignment operators are used to assigning values to the variables.

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

```
# Examples of Assignment Operators
a = 10

# Assign value
b = a
print(b)

# Add and assign value
b += a
print(b)

# Subtract and assign value
b -= a
print(b)

# multiply and assign
b *= a
print(b)

# bitwise lishift operator
b <<= a
print(b)
```

Output

10
20
10
100
102400

Identity Operators:

is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

is True if the operands are identical
is not True if the operands are not identical

```
a = 10
b = 20
c = a

print(a is not b)
print(a is c)
```

Output
True
True

Membership Operators

in and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

in True if value is found in the sequence
not in True if value is not found in the sequence

```
# Python program to illustrate
# not 'in' operator
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
    print("x is NOT present in given list")
else:
    print("x is present in given list")

if (y in list):
    print("y is present in given list")
else:
    print("y is NOT present in given list")
```

Output

```
x is NOT present in given list
y is present in given list
```

Arithmetic Expressions

You can combine operators to form complex expressions. An **expression** is a combination of numbers, operators, and parentheses that Python can compute, or **evaluate**, to return a value.

Python

```
>>> 2*3 - 1
5

>>> 4/2 + 2**3
10.0

>>> -1 + (-3*2 + 4)
-3
```

Class outline

1. Basic Mathematical Operations
2. **Attaching Names to Numbers**
3. **Different Kinds of Numbers**
4. Working with Fractions
5. Complex Numbers
6. Getting User Input
7. Handling Exceptions and Invalid Input

2. Attaching Names to Numbers

3. Different Kinds of Numbers

```
# adding two values
>>> 1 + 1
2
# setting a variable
>>> a = 1
>>> a
1
# checking a variables type
>>> type(a)
<type 'int'>
# an arbitrarily long integer
>>> a = 1203405503201
>>> a
1203405503201L
>>> type(a)
<type 'long'>
```

```
# real numbers
>>> b = 1.2 + 3.1
>>> b
4.2999999999999998
>>> type(b)
<type 'float'>
# complex numbers
>>> c = 2+1.5j
>>> c
(2+1.5j)
```

The four numeric types in Python on 32-bit architectures are:

- `integer` (4 byte)
- `long integer` (any precision)
- `float` (8 byte like C's double)
- `complex` (16 byte)

The numpy module, which we will see later, supports a larger number of numeric types.

CREATING STRINGS

```
# using double quotes
>>> s = "hello world"
>>> print s
hello world
# single quotes also work
>>> s = 'hello world'
>>> print s
hello world
```

STRING OPERATIONS

```
# concatenating two strings
>>> "hello " + "world"
'hello world'

# repeating a string
>>> "hello " * 3
'hello hello hello '
```

STRING LENGTH

```
>>> s = "12345"
>>> len(s)
5
```

SPLIT/JOIN STRINGS

```
# split space delimited words
>>> wrd_lst = s.split()
>>> print wrd_lst
['hello', 'world']

# join words back together
# with a space in-between
>>> ' '.join(wrd_lst)
hello world
```

Class outline

1. Basic Mathematical Operations
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. **Working with Fractions**
5. Complex Numbers
6. Getting User Input
7. Handling Exceptions and Invalid Input

4. Working with Fractions

Python can also handle fractions, it uses Python's fractions module. You can think of a module as a program written by someone else that you can use in your own programs. A module can include classes, functions, and even label definitions. It can be part of Python's standard library.

Fraction(numerator, denominator)

```
>>> from fractions import Fraction
>>> f = Fraction(3, 4)
>>> f
 Fraction(3,4)
```

```
>>> from fractions import *
>>> Fraction(0.75)
Fraction (3,4)
>>> Fraction ('3/4')
Fraction (3,4)
```

```
>>> Fraction(4) == Fraction(4,1)
True
```

```
>>> Fraction() == Fraction(0,1) == Fraction (0,3)
True

>>> Fraction(3,5) + Fraction (1,5)
Fraction(4, 5)

>>> Fraction(2,7) * Fraction (7,2)
Fraction(1, 1)
```

$$\frac{2}{7} \times \frac{7}{2} = \frac{14}{14} = 1 = \frac{1}{1}$$

```
>>> 49 ** Fraction(1,2)
7.0
```

$$49^{\frac{1}{2}} = \sqrt{49} = 7$$

```
>>> a = Fraction(2,3)
>>> b = Fraction (5,6)
>>> a + b, a - b, a * b, a / b, a//b
(Fraction(3, 2), Fraction(-1, 6), Fraction(5, 9), Fraction(4, 5), 0)
```

$$\frac{2}{3} + \frac{5}{6} = \frac{4}{6} + \frac{5}{6} = \frac{9}{6} = \frac{3}{2}$$

$$\frac{2}{3} - \frac{5}{6} = \frac{4}{6} - \frac{5}{6} = -\frac{1}{6}$$

$$\frac{2}{3} \times \frac{5}{6} = \frac{10}{18} = \frac{5}{9}$$

$$\frac{2}{3} \div \frac{5}{6} = \frac{2}{3} \times \frac{6}{5} = \frac{12}{15} = \frac{4}{5}$$

```
>>>Fraction(3, 4) + 1 + Fraction(1/4)
Fraction(7, 4)

>>> Fraction('-5e3')
Fraction(-5000, 1)

>>> Fraction(-1,5)
Fraction(-1, 5)

>>> Fraction(1,-5)
Fraction(-1, 5)



>>> a = 1.54657
>>> Fraction(a).limit_denominator(5)
Fraction(3, 2)
```

$$1.54657 = \frac{154657}{100000} \approx \frac{14}{9} \approx \frac{3}{2}$$

Class outline

1. Basic Mathematical Operations
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. Working with Fractions
5. **Complex Numbers**
6. Getting User Input
7. Handling Exceptions and Invalid Input
8. Writing Programs That Do the Math for You

5. Complex Numbers

- The numbers we've seen so far are the so-called real numbers. Python also supports complex numbers with the imaginary part identified by the letter j or J
- (as opposed to the letter i used in mathematical notation).

For example, the

- complex number 2 + 3i would be written in Python as 2 + 3j: >>> a = 2 + 3j
- >>> type(a)
- <class 'complex'> As you can see, when we use the type() function on a complex number, Python tells us that this is an object of type complex.

Python can also handle complex numbers and its associated functions using the file “cmath”. Complex numbers have their uses in many applications related to mathematics and python provides useful tools to handle and manipulate them.

Function:

Real()
Complex()
Imag()

Converting real numbers to complex number:

An complex number is represented by “ **x + yi** “. Python converts the real numbers x and y into complex using the function **complex(x,y)**. The real part can be accessed using the function **real()** and imaginary part can be represented by **imag()**.

```
# Python code to demonstrate the working of
# complex(), real() and imag()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

Class outline

1. Basic Mathematical Operations
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. Working with Fractions
5. Complex Numbers
6. **Getting User Input**
7. Handling Exceptions and Invalid Input

6. Getting User Input

Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

•input (prompt)

input ( ) : This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python. For example:

```
# Python program showing
# a use of input()

val = input("Enter your value: ")
print(val)
```

Typecasting the input to Integer:

```
# input
num1 = int(input())
num2 = int(input())
# printing the sum in integer
print(num1 + num2)
```

Typecasting the input to Float:

```
# input
num1 = float(input())
num2 = float(input())

# printing the sum in float
print(num1 + num2)
```

Typecasting the input to String:

```
# input
string = str(input())

# output
print(string)
```

Python

```python
# age.py

age = int(input("Enter your age: "))

if age >= 0 and age <= 9:
    print("You are a child!")
elif age > 9 and age <= 18:
    print("You are an adolescent!")
elif age > 18 and age <= 65:
    print("You are an adult!")
elif age > 65:
    print("Golden ages!")
```

Taking multiple inputs from user in Python

The developer often wants a user to enter multiple values or inputs in one line.

1- Using split() method
2- Using List comprehension

1- Using split() method

Syntax :

```
input().split(separator, maxsplit)
```

```
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
```

2- Using List comprehension:

List comprehension is an elegant way to define and create list in Python. We can create lists just like mathematical statements in one line only. It is also used in getting multiple inputs from a user.

```
# taking two input at a time
x, y = [int(x) for x in input("Enter two values: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)
print()
```

```
# taking three input at a time
x, y, z = [int(x) for x in input("Enter three values: ").split()]
print("First Number is: ", x)
print("Second Number is: ", y)
print("Third Number is: ", z)
```

```
# taking multiple inputs at a time
x = [int(x) for x in input("Enter multiple values: ").split()]
print("Number of list is: ", x)
```

Class outline

1. Basic Mathematical Operations
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. Working with Fractions
5. Complex Numbers
6. Getting User Input
7. **Handling Exceptions and Invalid Input**

7. Handling Exceptions and Invalid Input

Three types of Errors

Complie time Error

Syntax Error
E.g.
Missing (:)
Wrong Spelling

Logical Error

Wrong output

2+2= 5X

Run time Error

Your code completely complied and no error.
6/0 = ?

```
A = 5
B = 2
Print(A/B)
#Statement : Normal -- Normal statement, it will not give an error
```

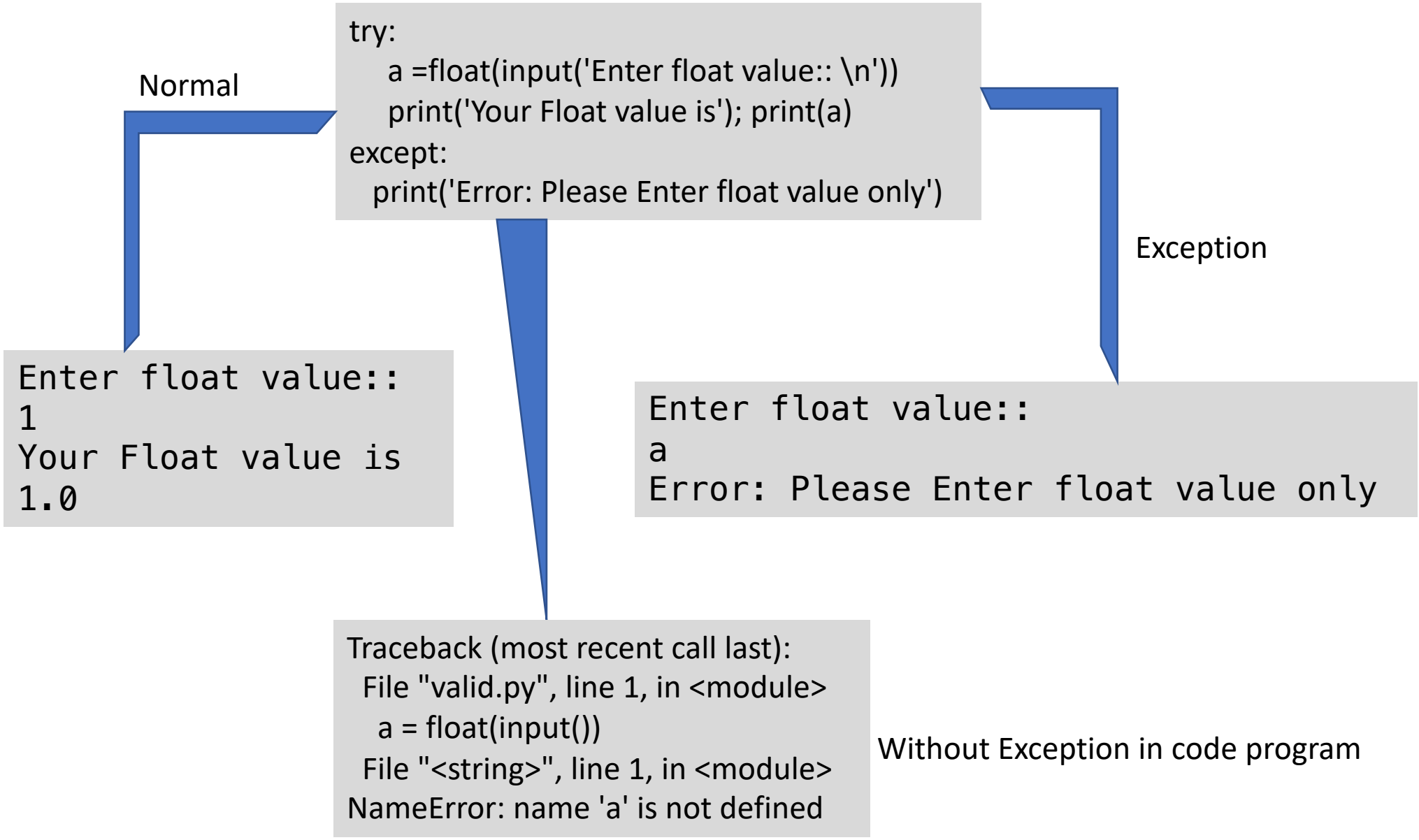
```
A = 5; B = 2 ; C = 0
Print (A/B)
Print(A/C)  #→ execution stop here, Critical Statement
```

Exception: description of the error

An exception is an **error** that happens during the execution of a program. Exceptions are known to non-programmers as instances that do not conform to a general rule. The name "exception" in computer science has this meaning as well: It implies that the problem (the exception) doesn't occur frequently, i.e. the exception is the "exception to the rule".

Exception handling is a construct in some programming languages to handle or deal with errors automatically. Many programming languages like C++, Objective-C, PHP, Java, Ruby, Python, and many others have built-in support for exception handling.

Error handling is generally resolved by saving the state of execution at the moment the error occurred and interrupting the normal flow of the program to execute a special function or piece of code, which is known as the exception handler. Depending on the kind of error ("division by zero", "file open error" and so on) which had occurred, the error handler can "fix" the problem and the programm can be continued afterwards with the previously saved data.


try:
a =float(input('Enter float value:: \n'))
print('Your Float value is'); print(a)
except:
print('Error: Please Enter float value only')
Normal
Exception
Enter float value::
1
Your Float value is
1.0
Enter float value::
a
Error: Please Enter float value only
Traceback (most recent call last):
File "valid.py", line 1, in <module>
a = float(input())
File "<string>", line 1, in <module>
NameError: name 'a' is not defined
Without Exception in code program

To get information about the error, we can access the Exception class instance that describes the exception by using for example:
except Exception as e:

```
try:
    print("test")
    # generate an error: the variable test is not defined
    print(test)
except Exception as e:
    print("Caught an exception:" + str(e))
```

```
test
Caught an exception:name 'test' is not defined
```

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
...
```

Class outline

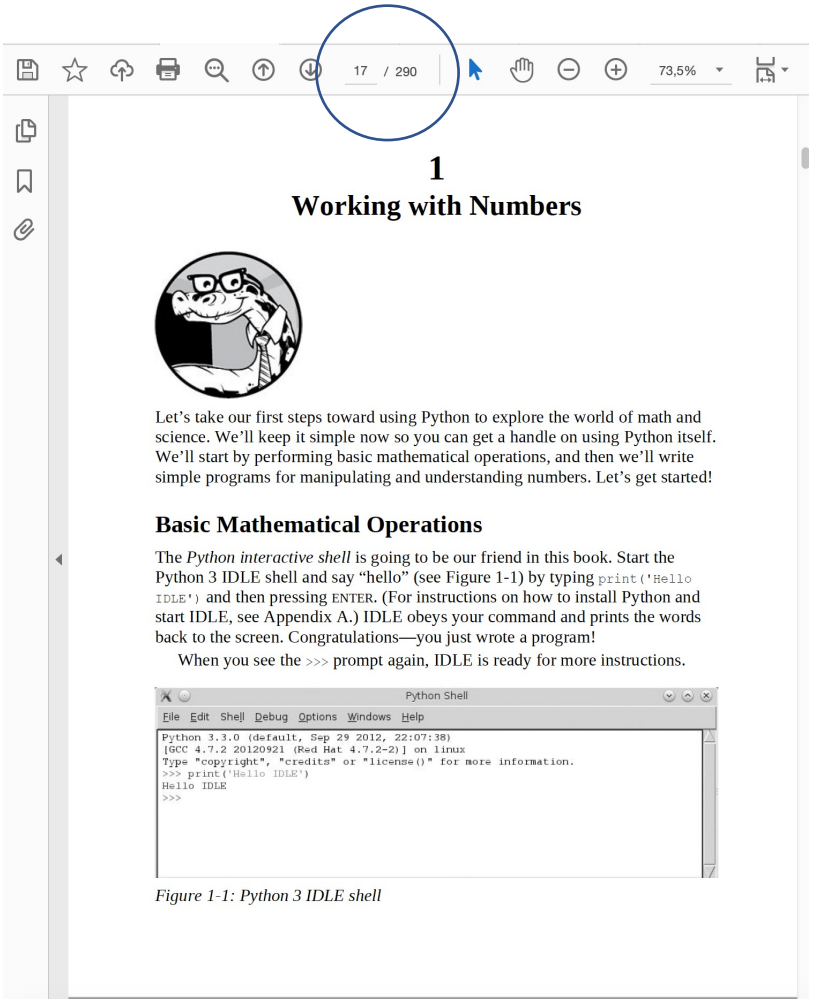
1. Basic Mathematical Operations
2. Attaching Names to Numbers
3. Different Kinds of Numbers
4. Working with Fractions
5. Complex Numbers
6. Getting User Input
7. Handling Exceptions and Invalid Input

Your Learnings

Assignment 1

- Read Pages (17 – 42) from the book:

DOING MATH WITH PYTHON Use Programming to Explore Algebra, Statistics, Calculus, and More! by Amit Saha



1
Working with Numbers

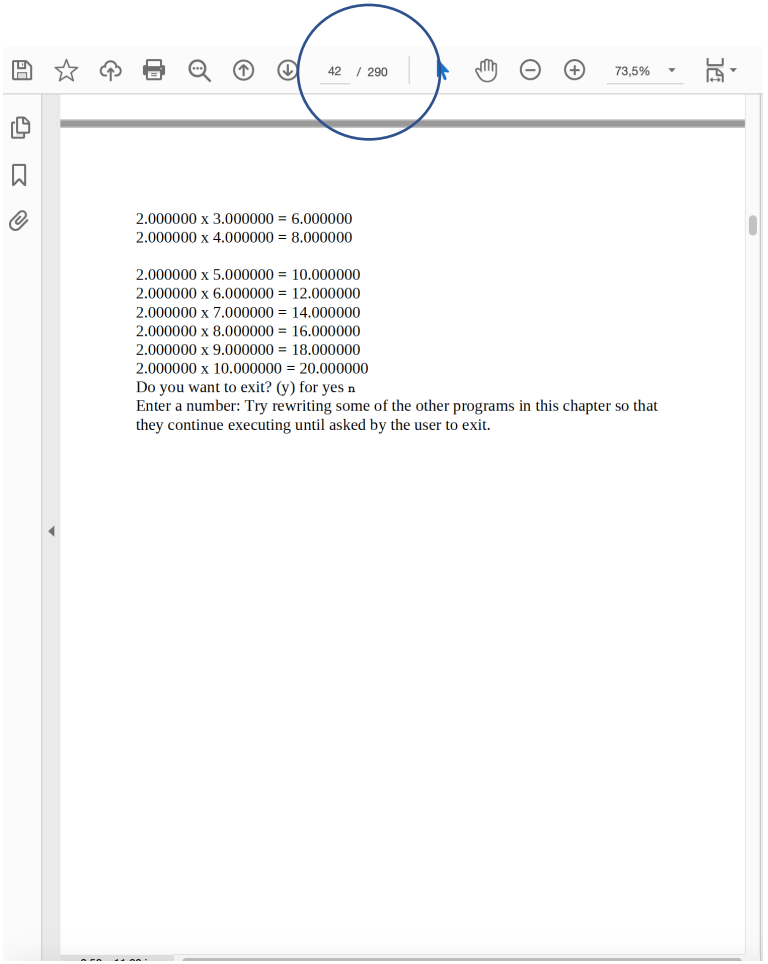
Let's take our first steps toward using Python to explore the world of math and science. We'll keep it simple now so you can get a handle on using Python itself. We'll start by performing basic mathematical operations, and then we'll write simple programs for manipulating and understanding numbers. Let's get started!

Basic Mathematical Operations

The *Python interactive shell* is going to be our friend in this book. Start the Python 3 IDLE shell and say "hello" (see Figure 1-1) by typing `print('Hello IDLE')` and then pressing ENTER. (For instructions on how to install Python and start IDLE, see Appendix A.) IDLE obeys your command and prints the words back to the screen. Congratulations—you just wrote a program!

When you see the `>>>` prompt again, IDLE is ready for more instructions.

Figure 1-1: Python 3 IDLE shell




```
2.000000 x 3.000000 = 6.000000
2.000000 x 4.000000 = 8.000000

2.000000 x 5.000000 = 10.000000
2.000000 x 6.000000 = 12.000000
2.000000 x 7.000000 = 14.000000
2.000000 x 8.000000 = 16.000000
2.000000 x 9.000000 = 18.000000
2.000000 x 10.000000 = 20.000000
Do you want to exit? (y) for yes n
Enter a number: Try rewriting some of the other programs in this chapter so that
they continue executing until asked by the user to exit.
```

End of Class 1