



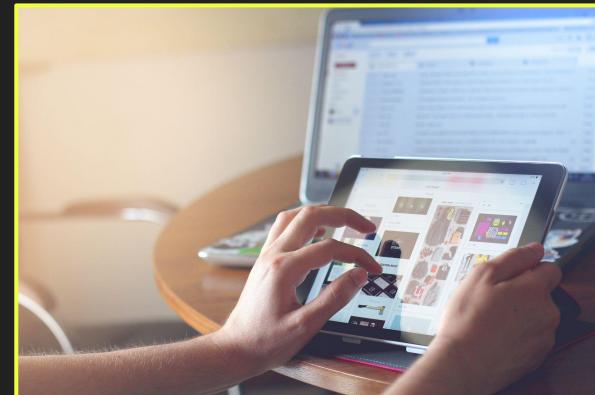
Introduction

IC152: Computing and Data Science

Rohit Saluja

Computing

Process of using computer technology to complete a given goal-oriented task.



- design and development of software and hardware systems
- structuring, processing and managing information
- pursuit of scientific studies
- making intelligent systems
- creating and using different media for entertainment and communication

Source: <https://www.techopedia.com/definition/6597/computing>

Image by fancycrave1 from Pixabay

Computing

- Cloud Computing (Services through Internet)
 - Google Vision API
- High Performance Computing (High Speed)
 - <https://iitmandi.ac.in/new/high-performance-cluster>
- Edge Computing (Compute at Edges: Latency)

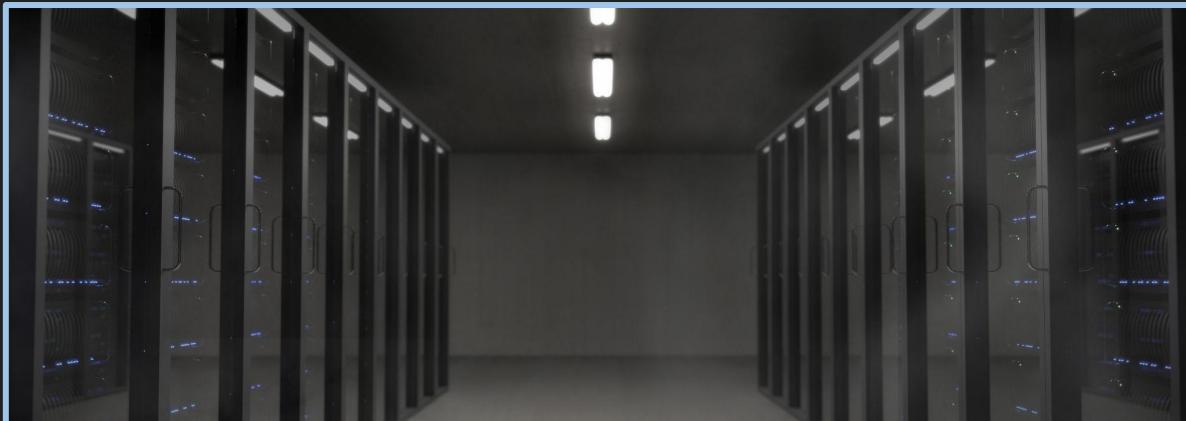


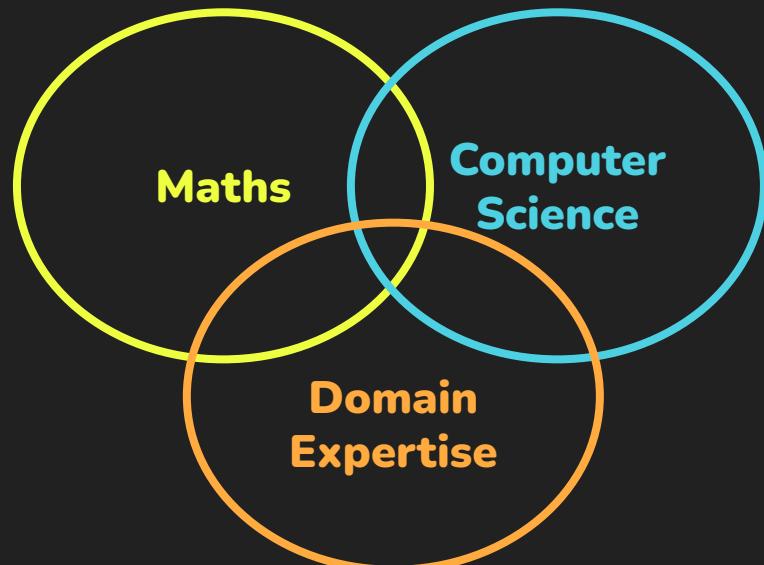
Image by Mudassar Iqbal and Elias from Pixabay

Data Science

Field of study that combines:

- Mathematics,
- Domain Expertise,
- and Programming

to extract meaningful insights from data.



Source: <https://www.datarobot.com/wiki/data-science/>

Image by Gerd Altmann from Pixabay

Binary Numbers



In Decimal Number System (dec ->10, i.e. base is 10):

- We have single digit numbers: 0, 1, 2 , 3, 4, 5, 6, 7, 8, 9
- We have units place, tens place, hundreds place etc.
- 123 means $3 \times 1 + 2 \times 10 + 1 \times 100$

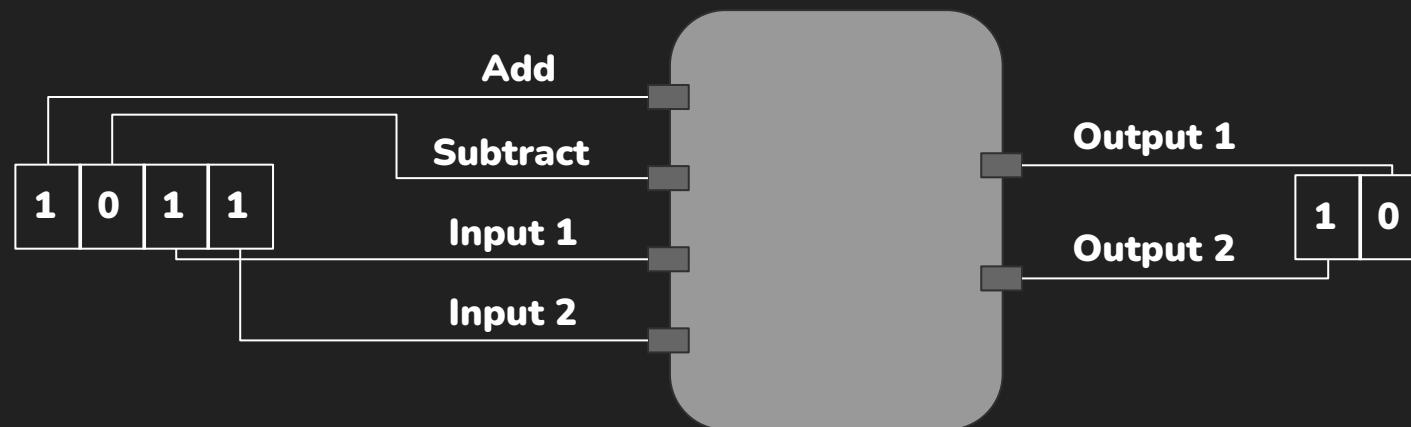
Similarly in Binary Number System (binary ->2, i.e. base is 2):

- We have single digit numbers: 0, 1
- You can say: We have units place, twos place, fours place etc.
- So 101 means $1 \times 1 + 0 \times 2 + 1 \times 4$ (= 5 in decimal number system)

Just like $9+1 = 10$ in decimal number system, we have $1+1 = 10$ in binary number system! (since 1 is the largest one digit in binary number system like 9 in decimal number system). Similarly: $1+0 = 1 = 0+1$, $10 + 01 = 11$, $011 + 010 = 101$.

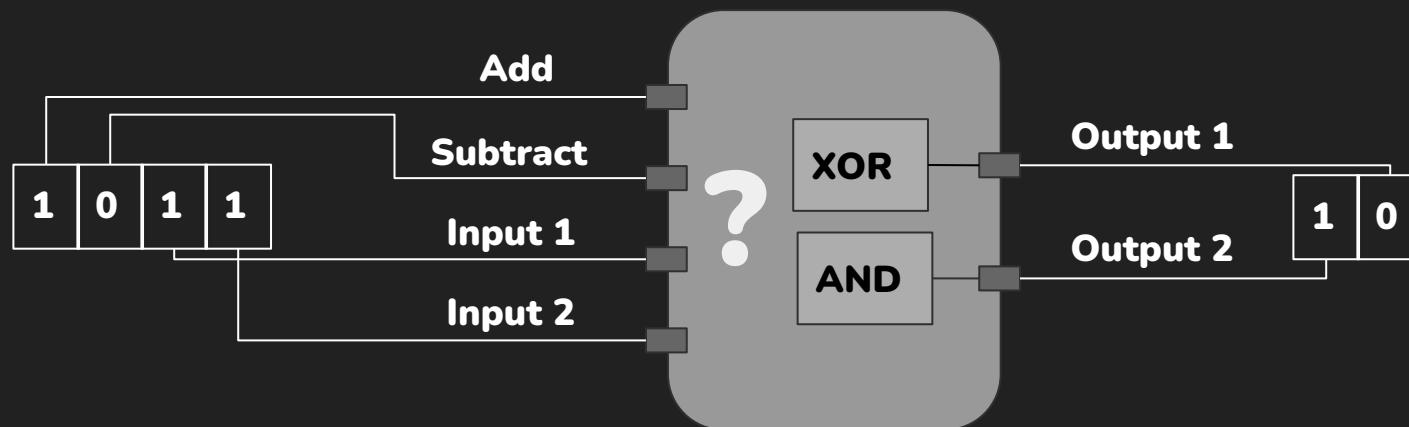
Machine Code

How a computer adds two one digit binary numbers?



Machine Code

How a computer adds two one digit binary numbers?



Programming Language

How about just saying this to your computer:-

Output = Input1 + Input2

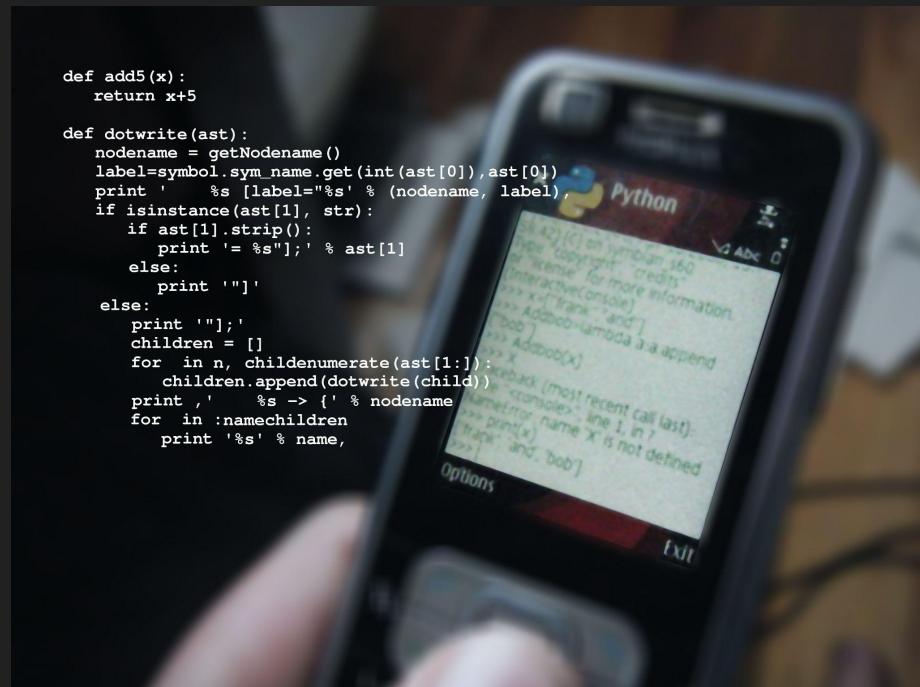
What will this Course Cover?

Python Basics

- Variables and Data Types:
Strings, Integers, Float
 - Array: Lists, Numpy
 - Operations, Operator Overloading
 - Tuples and Dictionaries
 - Functions: Inbuilt and User-defined
 - Conditional Statements and Loops
 - Errors: Syntax and Logical

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodeName()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '%s [%s=%s]' % (label,"is",nodename)
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= "%s"'; ' % ast[1]
        else:
            print '"'
    else:
        print ']'
    else:
        print '['
        children = []
        for n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ',%s-> {' % nodename
        for n in namechildren
            print '%s' % name,
```



What will this Course Cover?

Python Basics

- Reading and Writing Files
- Object Oriented Programming
- Command Line Arguments
- Handling Exceptions

Complexity Analysis

Stacks and Queues

```
class Bike:  
  
    def handle(self):  
        print("Used for acceleration.")  
  
    def break(self):  
        print("Used to stop bike.")
```

What will this Course Cover?

Plots: Visualizing Data

Statistics

Probability

Regression, interpolation

Machine learning

Clustering

Case studies, review



Some Real Life Applications

Tree Counting: <https://tinyurl.com/nd4msyf2>

GUI to annotate object boxes: <https://github.com/heartexlabs/labelImg>

- The above GUI can be modified to annotate Leaf Tips

Motorcycle Violations: <https://tinyurl.com/3x5hhtb3>

Character Control in Games: <https://github.com/sreyafrancis/PFNN>

- Video: <https://tinyurl.com/2d5a5k92>

OpenOCRCorrect: <https://tinyurl.com/47kabh28>

Evaluation

Theory (70%)

- Mid Sem: 30
- End Sem: 40

Lab (30%)

- Weekly Lab: 10
- Lab Exam: 20

Optional:

Project (0%)



Labs

Days: Monday/Wednesday/Thursday

Date and Time:

9 Nov'22 to 30 Nov'22, 17:30 to 19:30

1 Dec'22 to 26 Jan'23, 15:00 to 17:00

Labs: A10/A11

**Note: Only for 19 Nov'22 Lab will be on Saturday, 17:30 to 19:30
As we have No Instruction Day on 7 Nov'22**

Lab Schedule

Date	9 Nov'22	10 Nov'22	14 Nov'22	16 Nov'22	17 Nov'22	19 Nov'22
Day	Wednesday	Thursday	Monday	Wednesday	Thursday	Saturday
Time	17:30-19:30	17:30-19:30	17:30-19:30	17:30-19:30	17:30-19:30	17:30-19:30
Student IDs	B22001-22116	B22117-22232	B22233-22341	B22001-22116	B22117-22232	B22233-22341
Date	21 Nov'22	23 Nov'22	24 Nov'22	28 Nov'22	30 Nov'22	1 Dec'22
Day	Monday	Wednesday	Thursday	Monday	Wednesday	Thursday
Time	17:30-19:30	17:30-19:30	17:30-19:30	17:30-19:30	17:30-19:30	15:00-17:00
Student IDs	B22001-22116	B22117-22232	B22233-22341	B22001-22116	B22117-22232	B22233-22341

B22233-22341 includes B22233-22341 and 3-4 senior students throughout

Rescheduling Class

Option 1: Continue two classes till 1:20 p.m. with 5 minutes of break

Option 2: Extra Class on another Saturday

Project Ideas

Web page for pdf reader

Web page for text search in a document

Auto edit the short videos for social media platforms

Spell checker

GUI to annotate boxes around the objects in an image

Generate questions from a paragraph of text

Use grammar rules to split text in inflectional languages

Add captions to videos using Google's speech recognition API

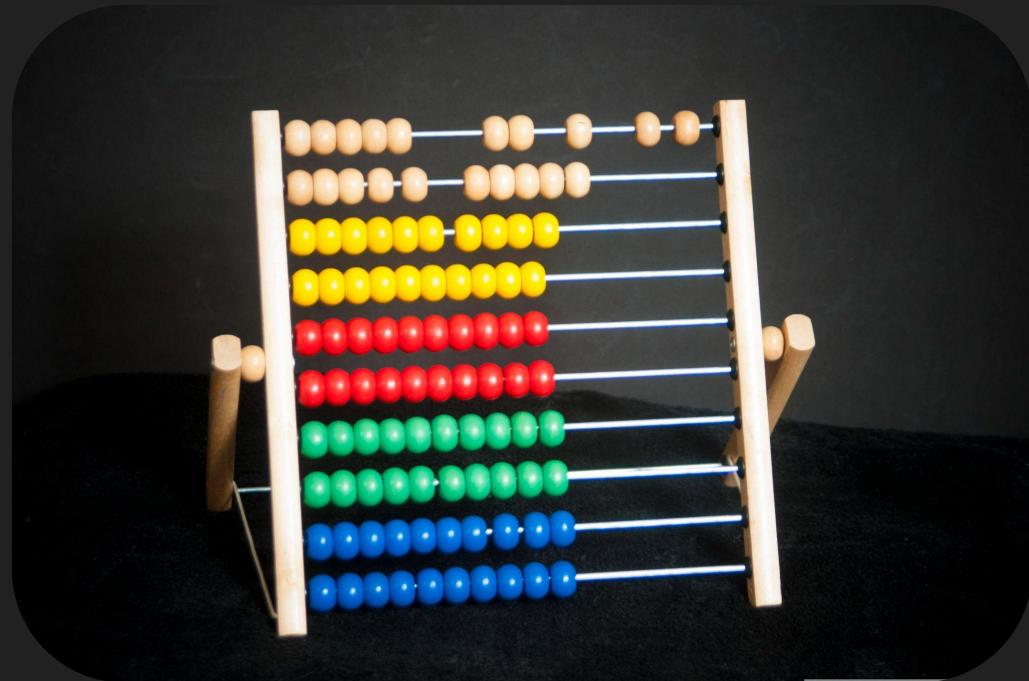
History of Computing Hardwares

Abacus (2700–2300 BC)

Principle: Counting (beads)

Operations:

- Addition
- Subtraction
- Multiplication
- Division
- Square Root
- Cubic Root



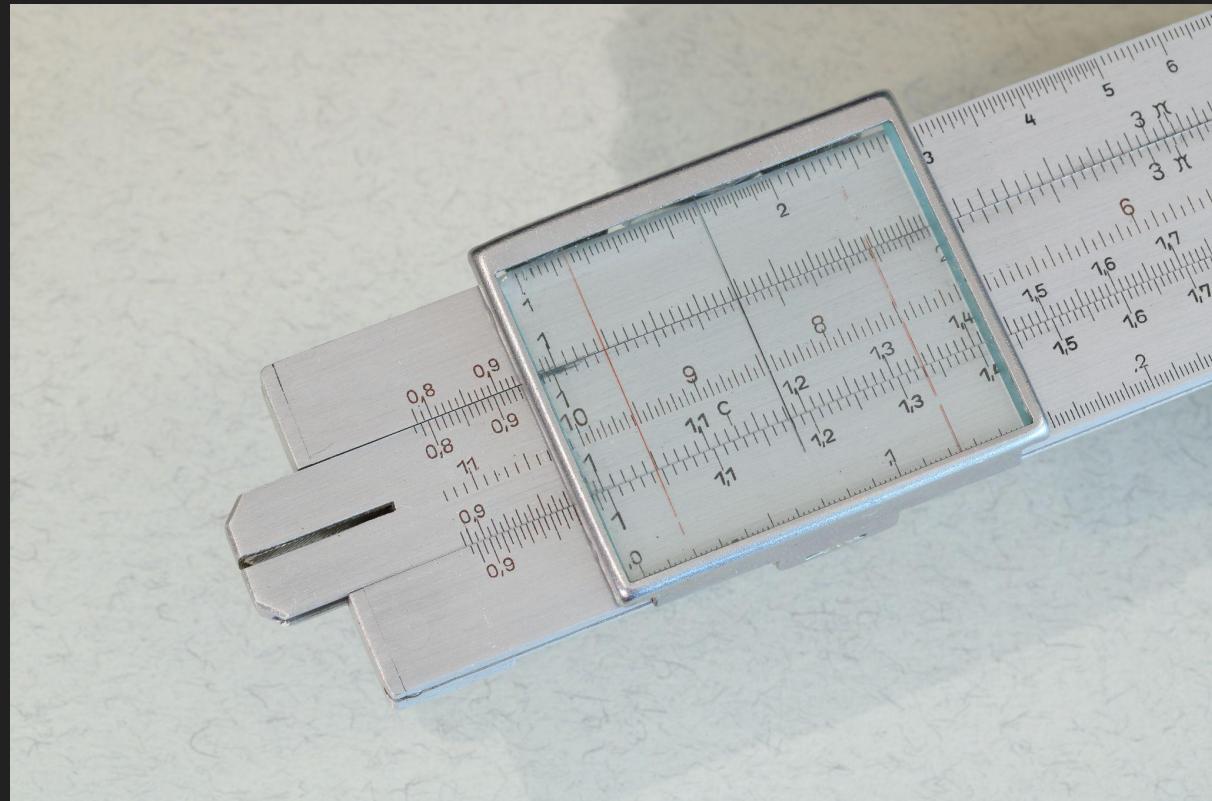
History of Computing Hardwares

Slide Rule (1814)

Principle: Log scales

Operations:

- Multiplication
- Division
- Square
- Square Root



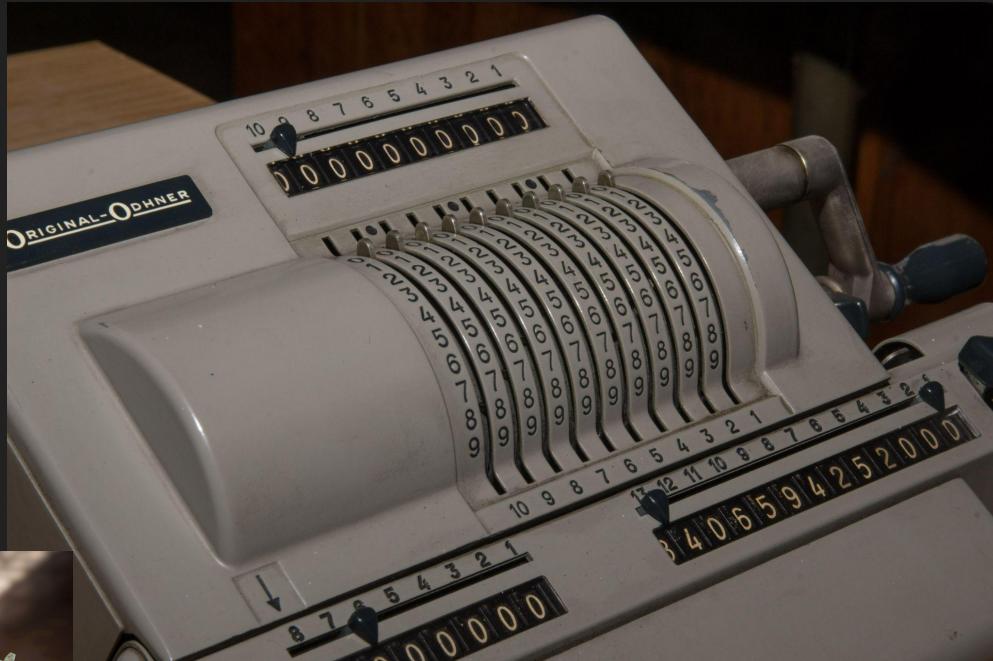
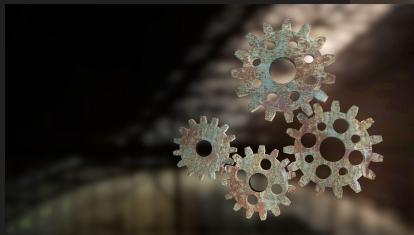
History of Computing Hardwares

Mechanical Calculators (1820)

Principle: Counting (gears)

Operations:

- Addition
- Subtraction
- Multiplication
- Division



Jacquard looms

- For Storage
- 1804
- Weaved patterns using punched cards
- punched card: paper that holds digital data represented by the presence or absence of holes in predefined positions.

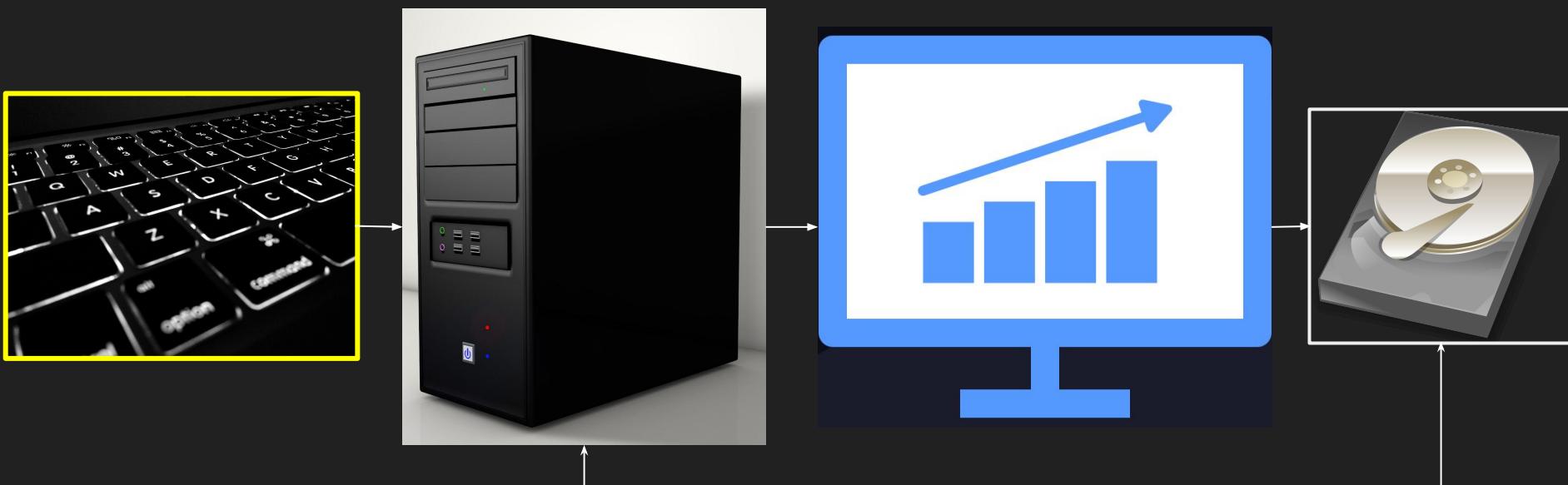


Image Source: https://en.wikipedia.org/wiki/Jacquard_machine

By User Ghw on en.wikipedia - Originally from en.wikipedia; description page is (was) here* 20:00, 28 July 2004 [[User:Ghw|Ghw]] 249×333 (64,550 bytes) (Close-up view of the punch cards used by Jacquard loom.); Public Domain, <https://commons.wikimedia.org/w/index.php?curid=1184856> https://en.wikipedia.org/wiki/Punched_card

Computer Basic Structure

- Computer: Electronic Device under Instruction and Programming Control
- Basic Operations: IPOS



Computer Basic Structure

Computer Types:

- General Purpose
 - Individual
 - Organizational
- Special Purpose

Examples?

Computer Basic Structure

Computer System Categories:

- **Hardware: Physical Components.**
- **Software: Enables hardware to perform set of tasks.**
 - Programming
 - Algorithms
 - Instructions

Computer Basic Structure

Software: Enables hardware to perform set of tasks

- System Software: Operation Coordination



Operating System



Utility Programs



Compiler, Interpreter

Computer Basic Structure

Software: Enables hardware to perform set of tasks

- Application Software: Performing Specific Task/Tasks



Text Processing



Math. Operations



DBMS

Analog and Digital Systems



Functional Units of Computer

- Input Unit: function is to read data/coded information from:-



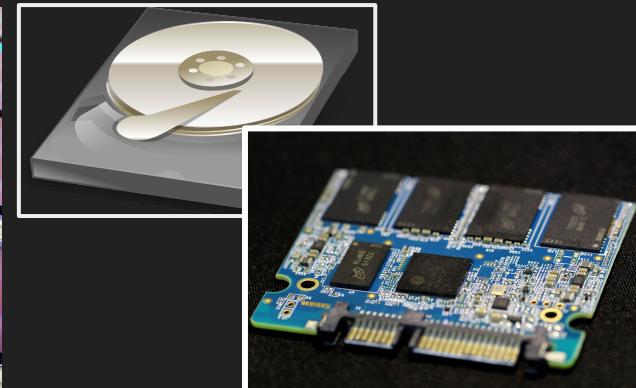
Functional Units of Computer

- Output Unit: function is to send/write processed data to display on:



Functional Units of Computer

- Storage Unit: function is to store, transfer, and extract: program and data
- Fix, reliable, and easy to find/fetch data



Floppy Disk: 1.44MB

Super Disk: 120/200/240 MB

Zip disk: 250 MB

Compact Disk (CD) 700MB

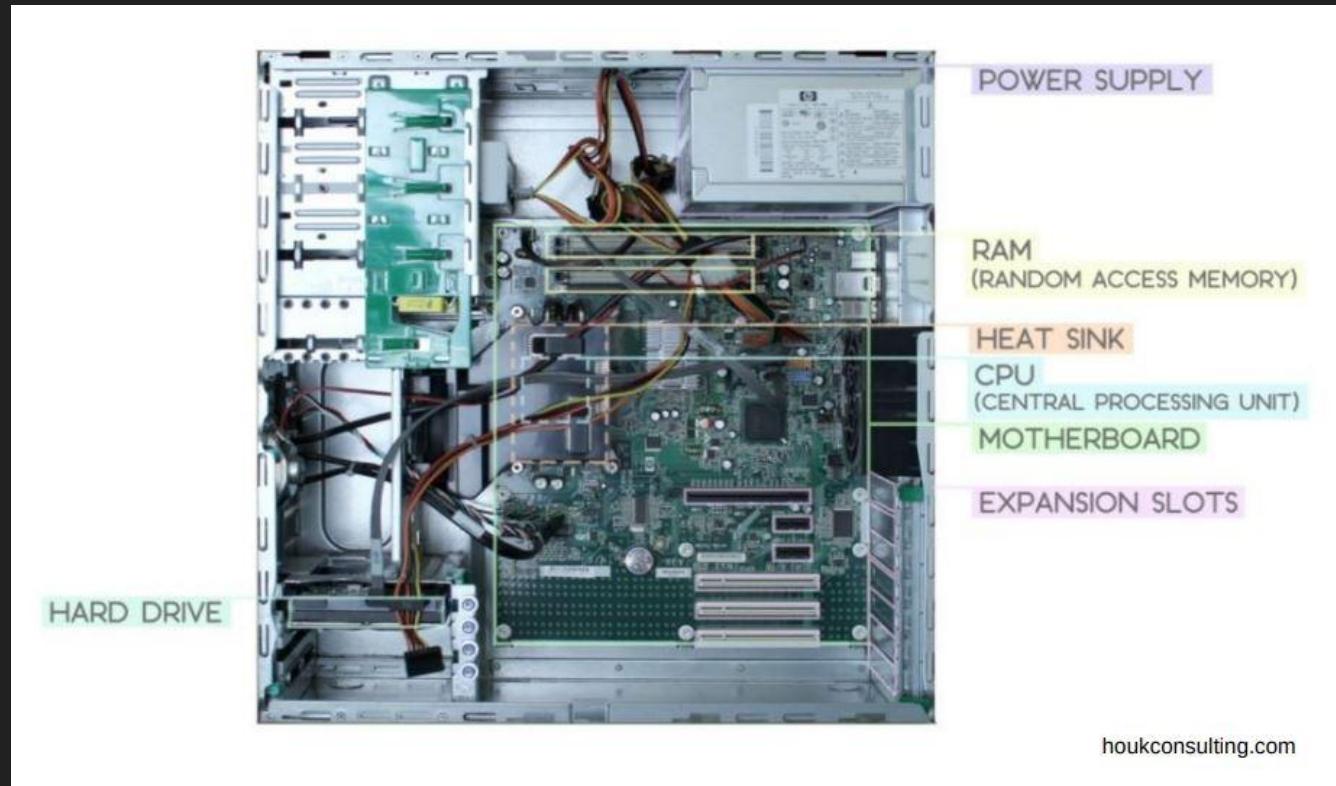
Digital Versatile Disk (DVD) 4.7GB

Hard Disk (HDD): 16GB-20TB

Solid State (SSD): 128GB-100TB

Functional Units of Computer

Processing Unit:
function is to
process
data/instructions



houkconsulting.com

Block Diagram

Registers Hold :

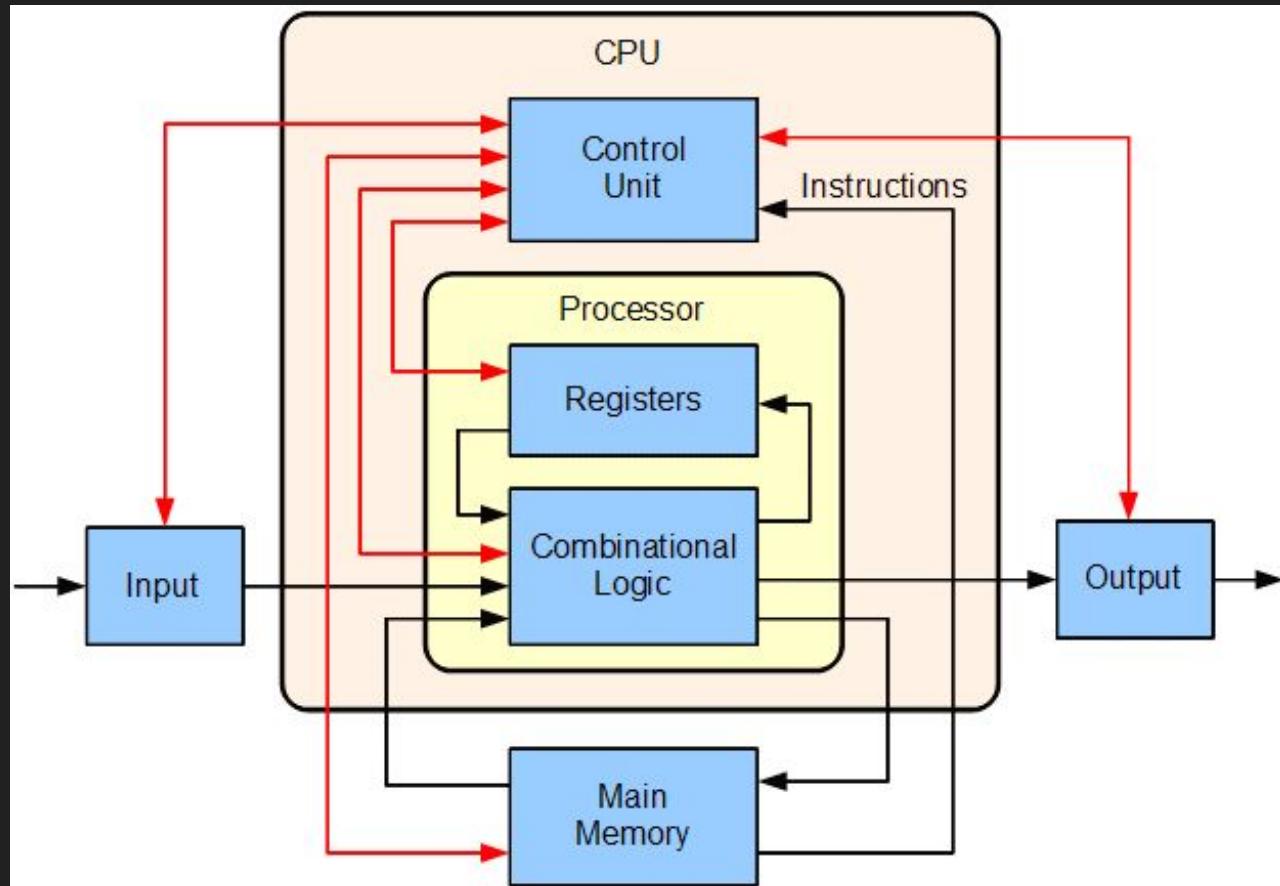
- Data
- Instructions
- Storage Address

Control Unit:

- Read instructions
- Send signals to other parts

Combinational Logic:

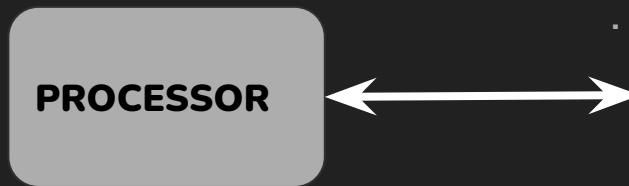
- Performs +/-
- Logical: AND/OR/XOR



The Computing Machine

Memory:

- Series of locations
- Store Information

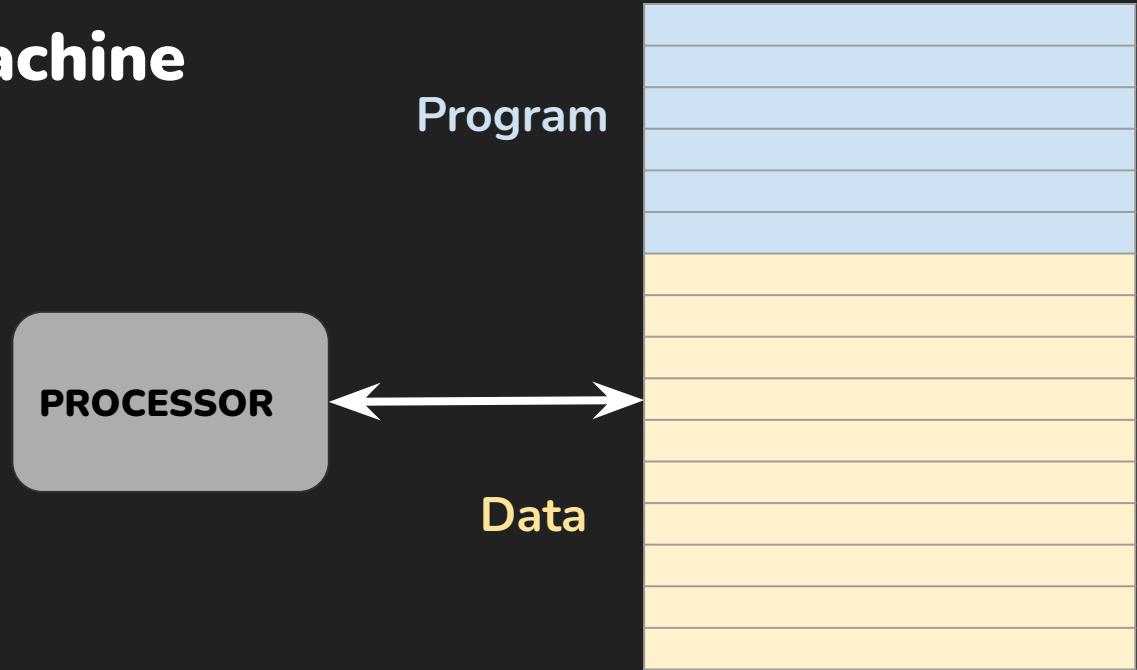


1 GIGABYTE

The Computing Machine

Memory:

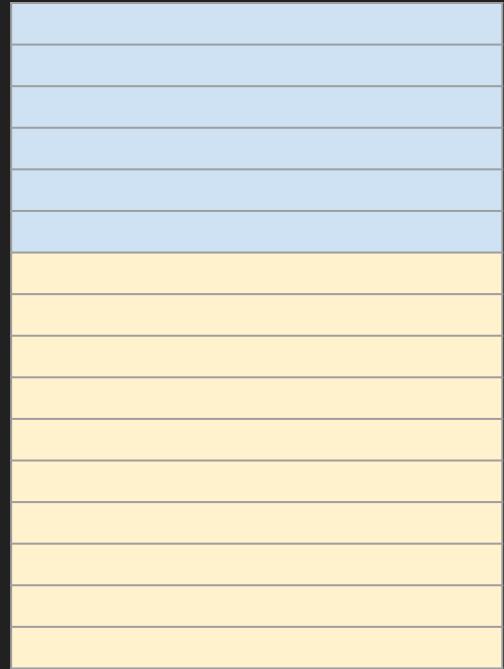
- Series of locations
- Store Information



The Computing Machine

Program

Data



The Stored Program von Neumann Computer

Program

Program

Instruction 1

1	0	1	1
0	1	1	0
.	.	.	.
1	0	1	0

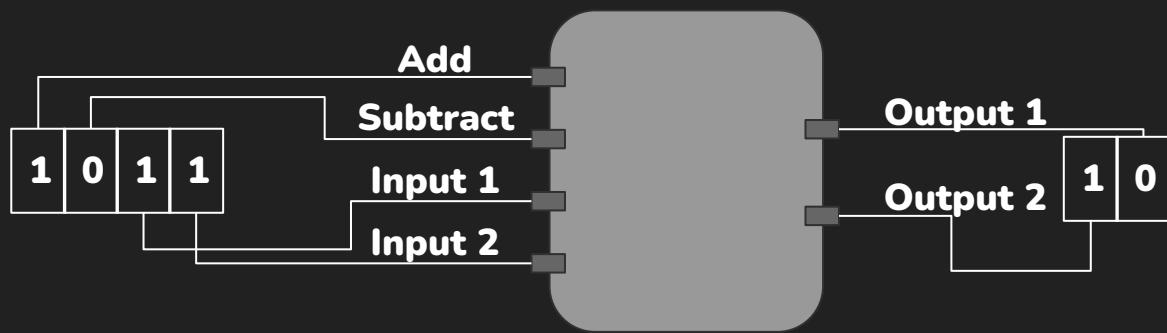
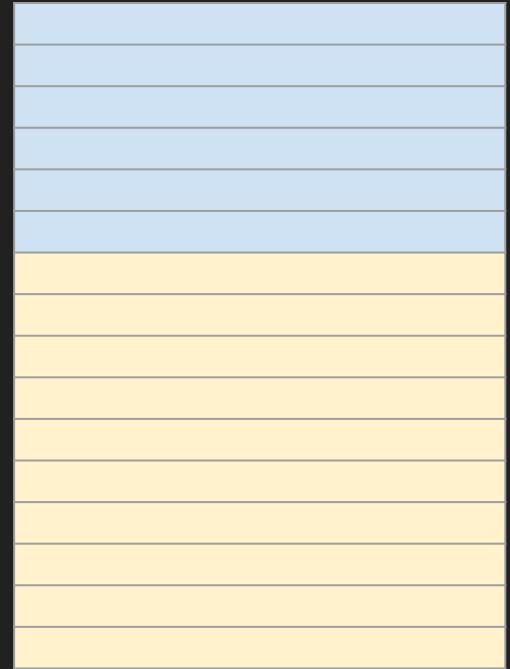
Instruction 2

.

.

.

Data



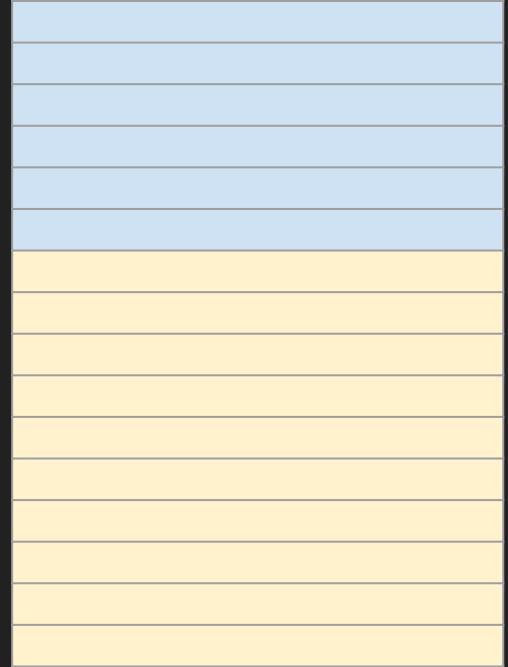
The Stored Program von Neumann Computer

Instructions:

- Operate on data
 - Program themselves can be treated as data
- Some may control the sequence of Instructions

Program

Data



Variables

Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer $x \neq 0$, s.t. $|x| < 1073741824$

ADDRESS	DATA	VARIABLE
1028	1	x
1032	52311	my_number
1036	- 534	myNumber
.	.	.
.	.	.
.	.	.

Variables

Memory location is given

ADDRESS

DATA

VARIABLE

- Name of a variable
- For programmer's ease

1028	1	x
------	---	---

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer $x \neq 0$, s.t. $|x| < 1073741824$

Variables

ADDRESS	DATA	VARIABLE
1028	1	x

Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer $x \neq 0$, s.t. $|x| < 1073741824$
- Object Overhead

```
>>> x = 1
>>> type(x)
<class 'int'>
```

Variables

Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer $x \neq 0$, s.t. $|x| < 1073741824$

ADDRESS	DATA	VARIABLE
1028	- 1073741823	x

```
>>> import sys  
>>> sys.getsizeof(0)  
24  
>>> sys.getsizeof(1)  
28  
>>> sys.getsizeof(1073741823)  
28  
>>> sys.getsizeof(1073741824)  
32  
>>> sys.getsizeof(-1073741823)  
28  
>>> bin(1073741823)  
'0b11111111111111111111111111111111'
```

Variables

Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer x s.t. $|x| < 1073741824$
- 1 byte (8 bits) for a character



```
>>> import sys  
>>> sys.getsizeof("")  
49  
>>> sys.getsizeof("c")  
50  
>>> sys.getsizeof("cd")  
51  
>>> |
```

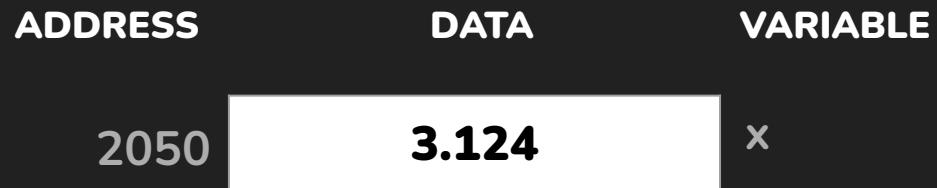
Variables

Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer x s.t. $|x| < 1073741824$
- 1 byte (8 bits) for a character
- 8 bytes (64 bits) for float



```
>>> x = 3.143
>>> import sys
>>> sys.getsizeof(x)
24
>>> sys.getsizeof(0.0)
24
>>> sys.getsizeof(12215245612621456121253212535
2.1235213)
24
>>> x = 122152456126214561212532125352.1235213
>>> x
1.2215245612621456e+29
```

Variables



Memory location is given

- Name of a variable
- For programmer's ease

Variable type

- Defines kind of data
- 4 bytes (32 bits) for an integer x s.t. $|x| < 1073741824$
- 1 byte (8 bits) for a character
- 8 bytes (64 bits) for float

$$n = (-1)^s \times 2^{e - 1023} \times (1+f)$$

$$s = 1$$

$$e = 1 \times 2^{10} + 0 \times 2^9 + \dots$$

$$f = 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + \dots$$

$$\begin{aligned} n &= -1 \times 2^{1024-1023} \times (1 + 0.5 + 0.25) \\ &= -3.5 \end{aligned}$$

Variables

Rules for variable names in python:

- Can only contain letters, numbers, and underscores.
- Can not start with a number

Tips:

- Descriptive: my_number for a number, myString for a string etc.
- Consistent: myNumber or my_number
- Traditions: avoid starting with _, start with lowercase
- keep them short

Bitwise Operations



Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & True	

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True
NOT	~	~ True	False

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True
NOT	~	~ True	False
Left Shift	<<	True << 1	
		True << 3	

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True
NOT	~	~ True	False
Left Shift	<<	True << 1	2
		True << 3	8

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True
NOT	~	~ True	False
Left Shift	<<	True << 1	2
		True << 3	8
Right Shift	>>	True >> 1	

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	True & False	False
OR		True False	True
XOR	^	True ^ False	True
NOT	~	~ True	False
Left Shift	<<	True << 1	2
		True << 3	8
Right Shift	>>	True >> 1	0

Bitwise Operations

```
>>> bin(2)
'0b10'
>>> bin(5)
'0b101'
>>> bin(7)
'0b111'
>>> bin(3)
'0b11'
>>> bin(-6)
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	2 & 3	

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	2 & 3	2

```
>>> bin(2)
'0b10'
>>> bin(5)
'0b101'
>>> bin(7)
'0b111'
>>> bin(3)
'0b11'
>>> bin(-6)
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7
XOR	^	$2 ^ 7$	5

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7
XOR	^	$2 ^ 7$	5
NOT	~	~ 5	

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7
XOR	^	$2 ^ 7$	5
NOT	~	~ 5	- 6

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7
XOR	^	$2 ^ 7$	5
NOT	~	~ 5	- 6
Left Shift	<<	$2 << 1$	4

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

Bitwise Operations

Operation	Python Operator	Example	Evaluates To
AND	&	$2 \& 3$	2
OR		$2 5$	7
XOR	^	$2 ^ 7$	5
NOT	~	~ 5	- 6
Left Shift	<<	$2 << 1$	4
Right Shift	>>	$5 >> 1$	2
		$5 >> 2$	1

```
>>> bin(2)  
'0b10'  
>>> bin(5)  
'0b101'  
>>> bin(7)  
'0b111'  
>>> bin(3)  
'0b11'  
>>> bin(-6)  
'-0b110'
```

1's Complement

- Signed representation
- Swap 0 with 1 and 1 with 0, and then append 1 on left
- $5 \rightarrow 101 \rightarrow \text{swap} \rightarrow 010 \rightarrow \text{append 1} \rightarrow 1010$
- Binary to Integer:

1	0	1	0
-8	4	2	1

1	1	0	1	0
-16	8	4	2	1

- $-8*1 + 4*0 + 2*1 + 1*0 = -6$
- Reverse of 1's complement:
- Remove 1 from left, and then swap 0 with 1 and 1 with 0,
- $\sim\sim 5 = 5$

Functions

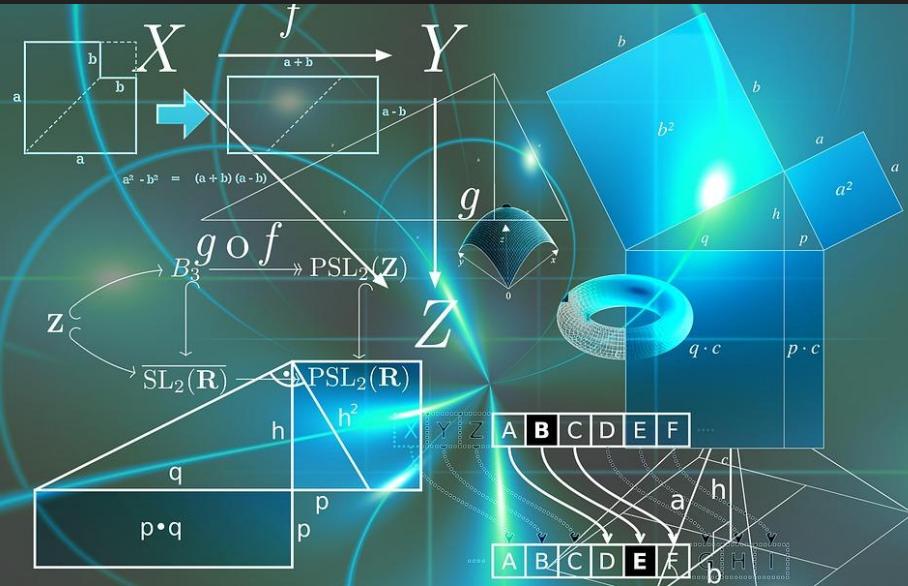


Image by [Gerd Altmann](#) from [Pixabay](#)

Functions

- Group of related statements
- Each function perform specific task
- Modularity
 - Examples:
 - Creating your own calculator in python: you need to define functions like add(), subtract(), multiply(), divide(), power(), etc.
 - A particular type of boxer first takes a step back and then punch.

Functions

E.g.

- print(), input()

```
>>> print("hello world")
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number|
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number10
>>> |
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number10
>>> print(N)
```

Functions

E.g.

- print(), input()

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number10
>>> print(N)
10
```

Functions

- arguments
 - parentheses
 - separated by commas
- output
- printing multiple arguments in same line and auto conversions

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number10
>>> |
```

```
>>> print("value of N is: ", N, "python can also convert floating point to string", 3.14)
```

Functions

- arguments
 - parentheses
 - separated by commas
- output
- printing multiple arguments in same line and auto conversions

```
>>> print("hello world")
hello world
>>> N = input("give me a number")
give me a number10
>>> |
```

```
>>> print("value of N is: ", N, "python can also convert floating point to string", 3.14)
value of N is: 10 python can also convert floating point to string 3.14
```

Variables

- Convert Functions

Function Call	Output
<code>float("3.14")</code>	3.14
<code>int("41")</code>	41
<code>str(12.312)</code>	'12.312'

Functions

User-defined

```
#userDefined.NewLine  
#prints a new line between two lines  
#using user defined function: newLine()  
#written by user name on dd/mm/yyyy
```

```
def newLine():  
    print()
```

}

function

```
print('first line')  
newLine()  
print('second line')
```

}

main program

```
first line
```

```
second line
```

Functions

User-defined
threeLines()

```
#userDefinedThreeLines
#prints three lines between two lines
#using user defined function: newLine()
#written by user name on dd/mm/yyyy

def newLine():
    print()

def threeLines():
    newLine()
    newLine()
    newLine()

print('first line')
threeLines()
print('second line')
```

first line

second line

Functions

User-defined
threeLines()
anyLines()

```
#userDefinedAnyLines
#prints any number lines between two lines
#using user defined function: newLine()
#written by user name on dd/mm/yyyy

def newLine():
    print('*')

def anyLines(n):
    for i in range(0,n):
        newLine()

    print('first line')
anyLines(4)
    print('second line')
```

```
first line
*
*
*
*
second line
```

String Methods

A screenshot of a Google search results page. The search query "transliterate to english" is entered in the search bar. Below the search bar, there are navigation links for All, Videos, News, Images, Books, More, and Tools. The "All" link is highlighted with a blue underline. It also indicates there are about 71,00,000 results found in 0.50 seconds. Below the search bar, there is a language translation interface with "Hindi" on the left and "English" on the right, separated by a double-headed arrow. Underneath this, the Hindi phrase "आई एम अ प्रोग्रामर" is shown, along with its transliteration "aaee aim a prograamar" and a suggestion "Did you mean: आई एम अ प्रोग्राम?". To the right, the English phrase "I am a programmer" is displayed. At the bottom of the search results, there are two small audio player icons.

transliterate to english

All Videos News Images Books More Tools

About 71,00,000 results (0.50 seconds)

Hindi English

आई एम अ प्रोग्रामर × I am a programmer

aaee aim a prograamar

Did you mean: आई एम अ प्रोग्राम?

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'

String Methods

- Are similar to functions as they have parenthesis in the end
- Return a string
- Must be called along with a string
- May or may not involve arguments



String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	'anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	'anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'
thought.swapcase()	

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	'anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'
thought.swapcase()	'aNYONE CAN LEARN PYTHON, PYTHON IS SO EASY. bEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	'anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'
thought.swapcase()	'aNYONE CAN LEARN PYTHON, PYTHON IS SO EASY. bEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.capitalize()	'Anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
thought.upper()	'ANYONE CAN LEARN PYTHON, PYTHON IS SO EASY. BEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.lower()	'anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'
thought.swapcase()	'aNYONE CAN LEARN PYTHON, PYTHON IS SO EASY. bEST WAY TO LEARN PYTHON IS USING IT FOR DIFFERENT TASKS.'
thought.capitalize()	'Anyone can learn python, python is so easy. best way to learn python is using it for different tasks.'
thought.title()	'Anyone Can Learn Python, Python Is So Easy. Best Way To Learn Python Is Using It For Different Tasks.'

String Methods

```
thought = " \tAnyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks. \n"
```

Method Call	Output
thought.strip()	'Anyone can learn python, python is so easy. Best way to learn python is using it for different tasks.'
thought.strip("\n")	' \tAnyone can learn python, python is so easy. Best way to learn python is using it for different tasks. '
thought.rstrip()	' \tAnyone can learn python, python is so easy. Best way to learn python is using it for different tasks.'

String Methods

```
thought = "Anyone can learn python, python is so easy. Best way to learn python is using it  
for different tasks."
```

Method Call	Output
<code>thought.replace('python', 'coding')</code>	'Anyone can learn coding, coding is so easy. Best way to learn coding is using it for different tasks.'
<code>thought.replace('python', 'coding',2)</code>	'Anyone can learn coding, coding is so easy. Best way to learn python is using it for different tasks.'
<code>thought.find('python')</code>	17
<code>thought.index('python')</code>	
<code>thought.count('python')</code>	3

String Methods

```
>>> thought = "Anyone can learn python, python is so easy. Best way to learn python is using it for different tasks."
```

```
>>> newThought = thought.replace('python', 'coding')
```

```
>>> print(newThought)
```

Anyone can learn coding, coding is so easy. Best way to learn coding is using it for different tasks.

```
>>> print(upper())
```

Traceback (most recent call last):

```
  File "<pyshell#36>", line 1, in <module>
```

```
    print(upper())
```

NameError: name 'upper' is not defined

String Operators

Concatenating Strings

Repeating Strings

```
>>> print("+ adds two numbers in python" + " but also concatenates two" + " or more strings.")  
+ adds two numbers in python but also concatenates two or more strings.  
>>> print("I want to emphasize again and again:  
" + "use interactive mode as calculator... "*3)  
I want to emphasize again and again: use interactive mode as calculator... use interactive mode as calculator... use interactive mode as calculator...
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> print(thought[0:10], "<-slice1, slice2->", thought[:10])  
Anyone can <-slice1, slice2-> Anyone can
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17  
>>> thought[0:17+len('python')]
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17  
>>> thought[0:17+len('python')]  
'Anyone can learn python'
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17  
>>> thought[0:17+len('python')]  
'Anyone can learn python'  
>>> thought[::-1]  
'ysae .tseB ot yaw nrael si nohtyp ,nohtyp enoynA si nohtyp'
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17  
>>> thought[0:17+len('python')]  
'Anyone can learn python'  
>>> thought[::-1]  
'.sksat tnereffid rof ti gnisu si nohtyp nrael ot yaw tseB .ysae  
os si nohtyp ,nohtyp nrael nac enoynA'  
>>> thought[17:]  
'python, python is so easy. Best way to learn python is using it  
for different tasks.'
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought.find('python')  
17  
>>> thought[0:17+len('python')]  
'Anyone can learn python'  
>>> thought[::-1]  
'.sksat tnereffid rof ti gnisu si nohtyp nrael ot yaw tseB .ysae  
os si nohtyp ,nohtyp nrael nac enoynA'  
>>> thought[17:]  
'python, python is so easy. Best way to learn python is using it  
for different tasks.'  
>>> thought[-6:]  
'tasks.'
```

String Slicing

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks."
```

```
>>> thought[17:17+len('python')] = "code"  
Traceback (most recent call last):  
  File "<pyshell#21>", line 1, in <module>  
    thought[17:17+len('python')] = "code"  
TypeError: 'str' object does not support item assignment
```

Strings are immutable data types in python

i.e. their values cannot be updated

So, how can we replace first and third occurrence of **python** in above string?

Membership Operators

```
>>> "a" in "abc"
```

```
True
```

```
>>> "A" in "abc"
```

```
False
```

```
>>> "ab" in "abc"
```

```
True
```

```
>>> "bc" in "abc"
```

```
True
```

```
>>> "bc" not in "abc"
```

```
False
```

Strings can be compared

```
>>> x = "cat" < "rat"
```

Strings can be compared

```
>>> x = "cat" < "rat"  
>>> print(x, type(x))
```

Strings can be compared

```
>>> x = "cat" < "rat"  
>>> print(x, type(x))  
True <class 'bool'>
```

Strings can be compared

```
>>> x = "cat" < "rat"  
>>> print(x, type(x))  
True <class 'bool'>
```

```
>>> myString = "12321"
```

Strings can be compared

```
>>> x = "cat" < "rat"  
>>> print(x, type(x))  
True <class 'bool'>
```

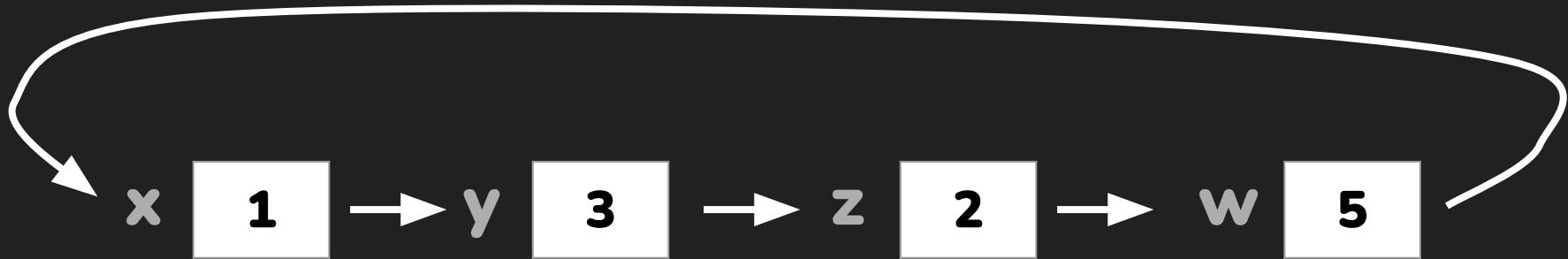
```
>>> myString = "12321"  
>>> myString == myString[::-1]
```

Strings can be compared

```
>>> x = "cat" < "rat"  
>>> print(x, type(x))  
True <class 'bool'>
```

```
>>> myString = "12321"  
>>> myString == myString[::-1]  
True
```

Swapping Numbers



Swap 2 numbers

Swap/exchange the values of variables x and y:

x 45 y -12



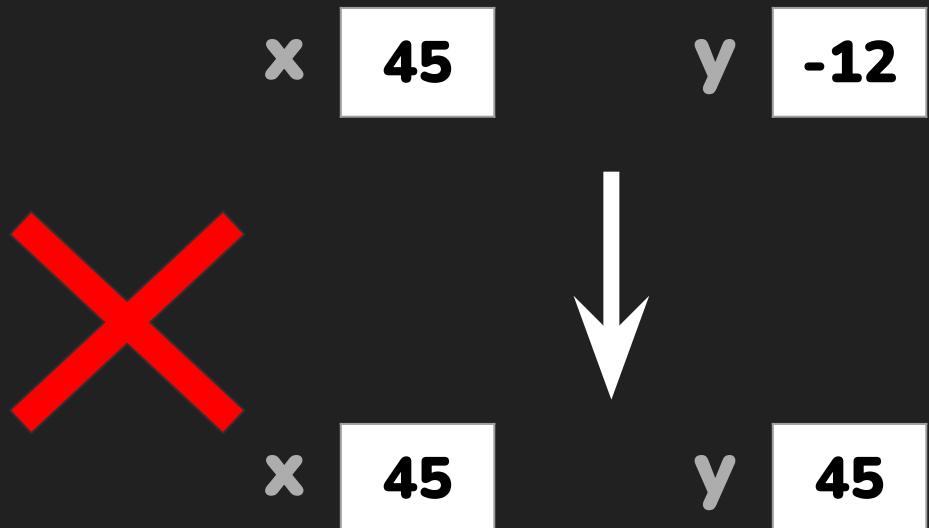
x -12 y 45

Swap 2 numbers

Swap/exchange the values of variables x and y:

Algorithm 1:

- 1) Let x and y be integers
- 2) Read values of x and y from user
- 2) Assign value of x to y
- 3) Assign value of y to x

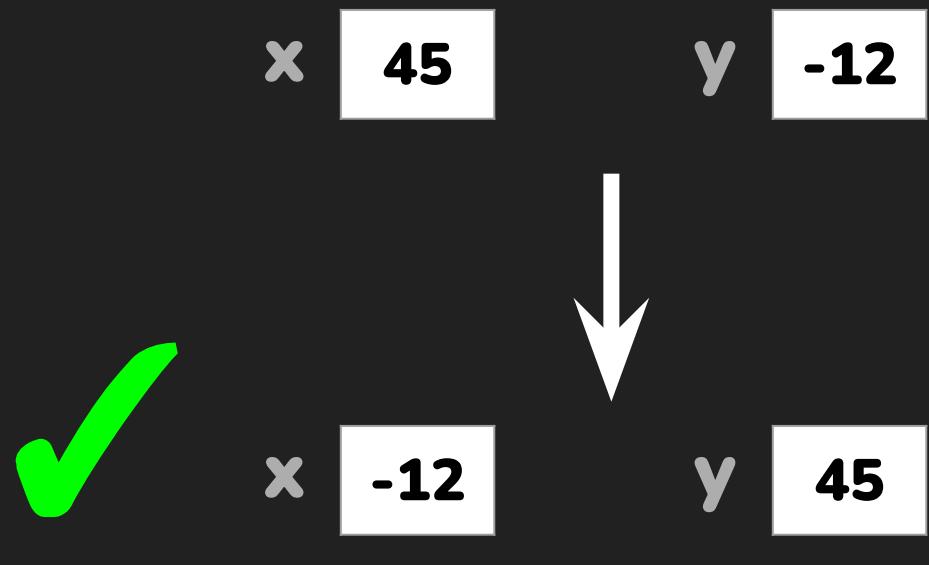


Swap 2 numbers

Swap/exchange the values of variables x and y:

Algorithm 2:

- 1) Let x and y be integers
- 2) Read values of x and y from user
- 3) Create temporary variable t
- 4) Assign value of x to t
- 5) Assign value of y to x
- 6) Assign value of t to y



Swap 2 numbers

Swap/exchange the values of variables x and y:

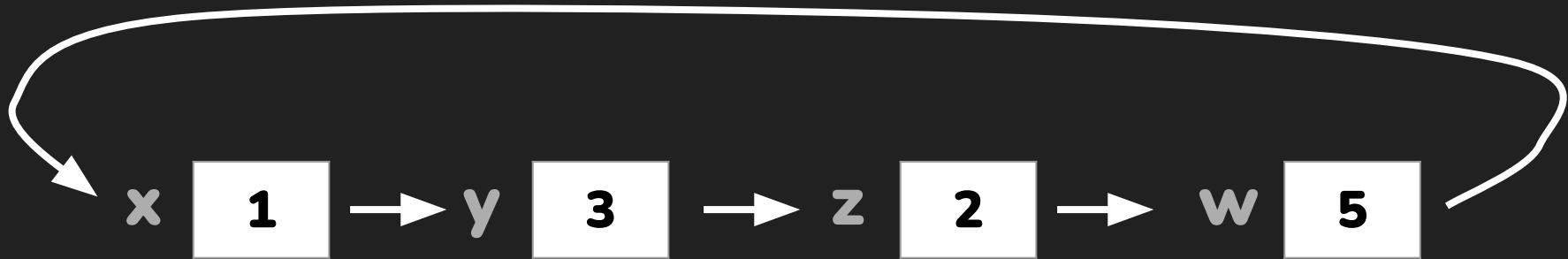
Algorithm 2:

- 1) Let x and y be integers**
- 2) Read values of x and y from user**
- 3) Create temporary variable t**
- 4) Assign value of x to t**
- 5) Assign value of y to x**
- 6) Assign value of t to y**

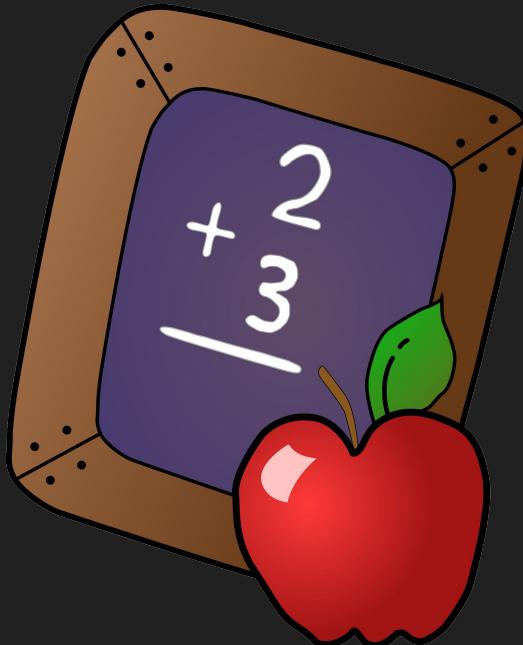
Algorithm:

- Set of rules/instructions
- Followed in calculations/problem solving/performing computation
- Clear
- Inputs
- Outputs
- Finite
- Feasible
- Language Independent
- Deterministic and Effective
- PCIOS

Swapping Numbers



Adding N Numbers



552

+

12

+

103

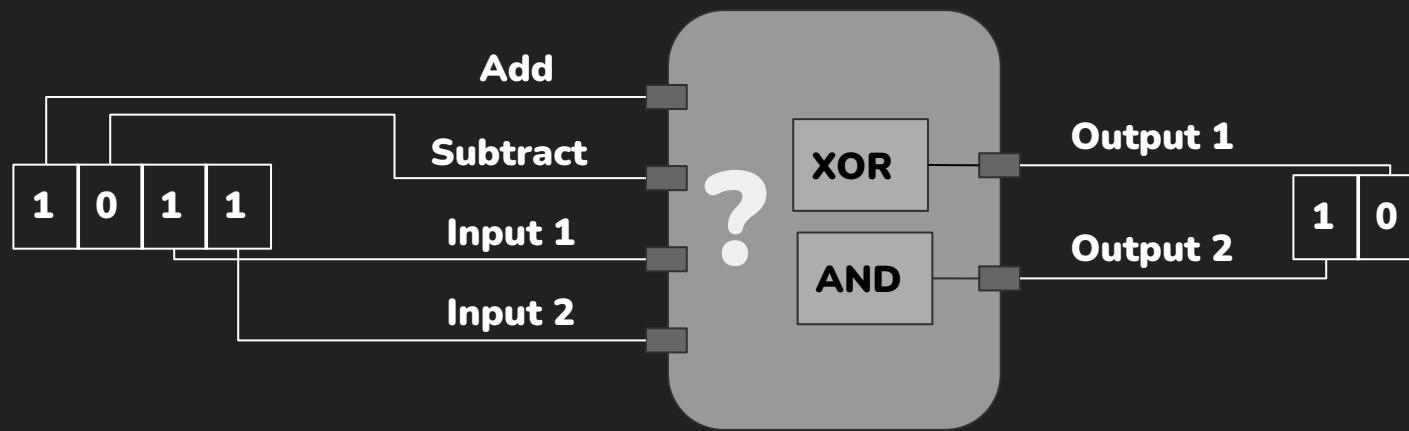
+

•

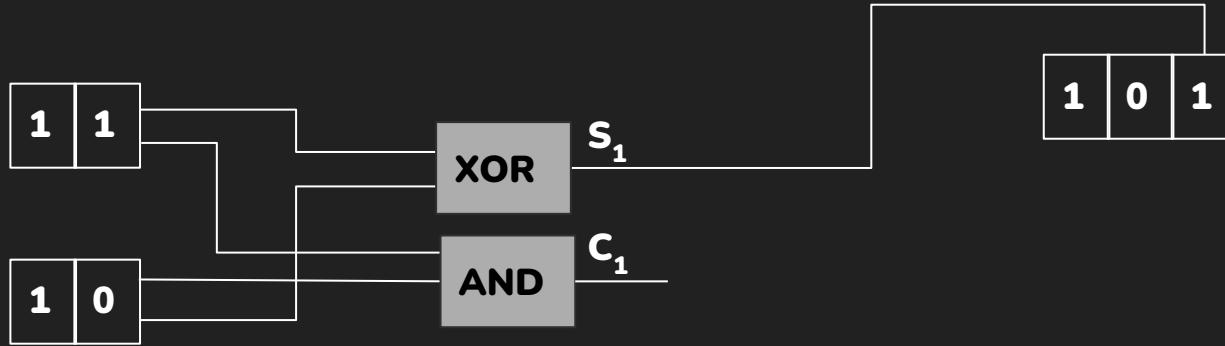
•

•

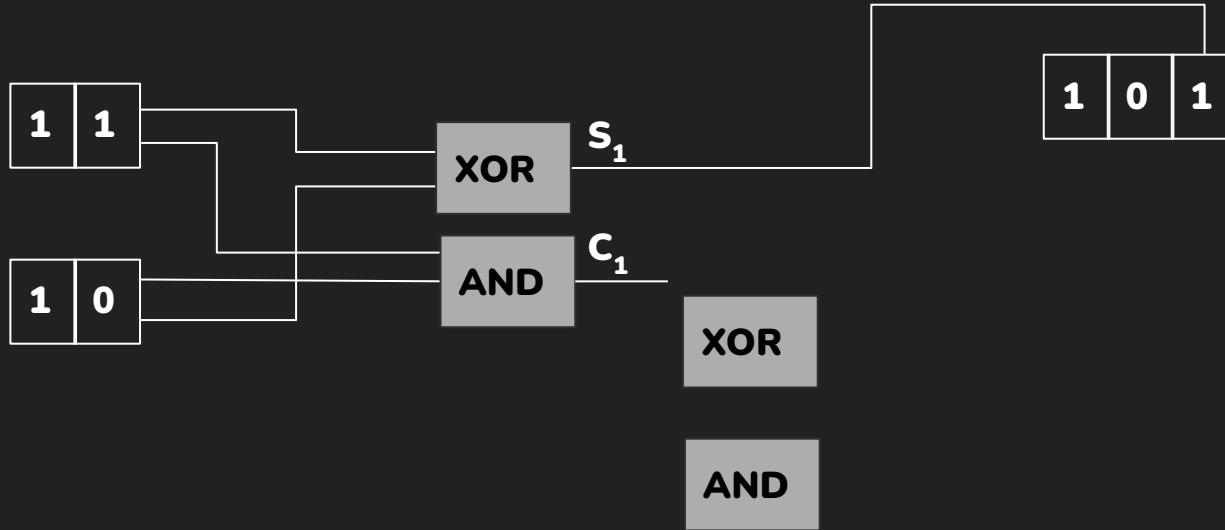
Adding 2 Numbers



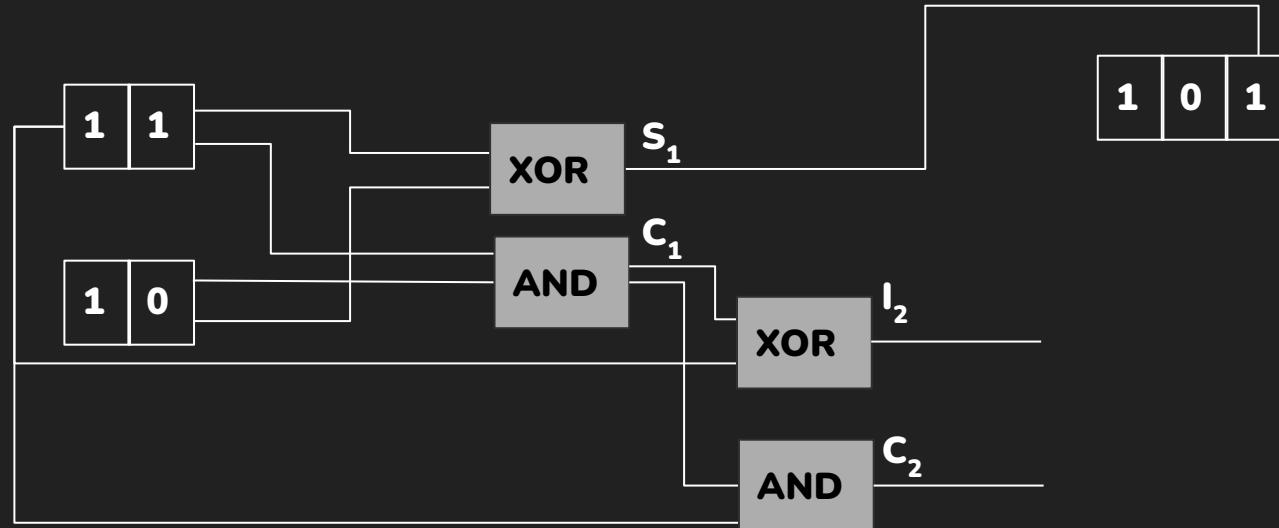
Adding 2 Numbers



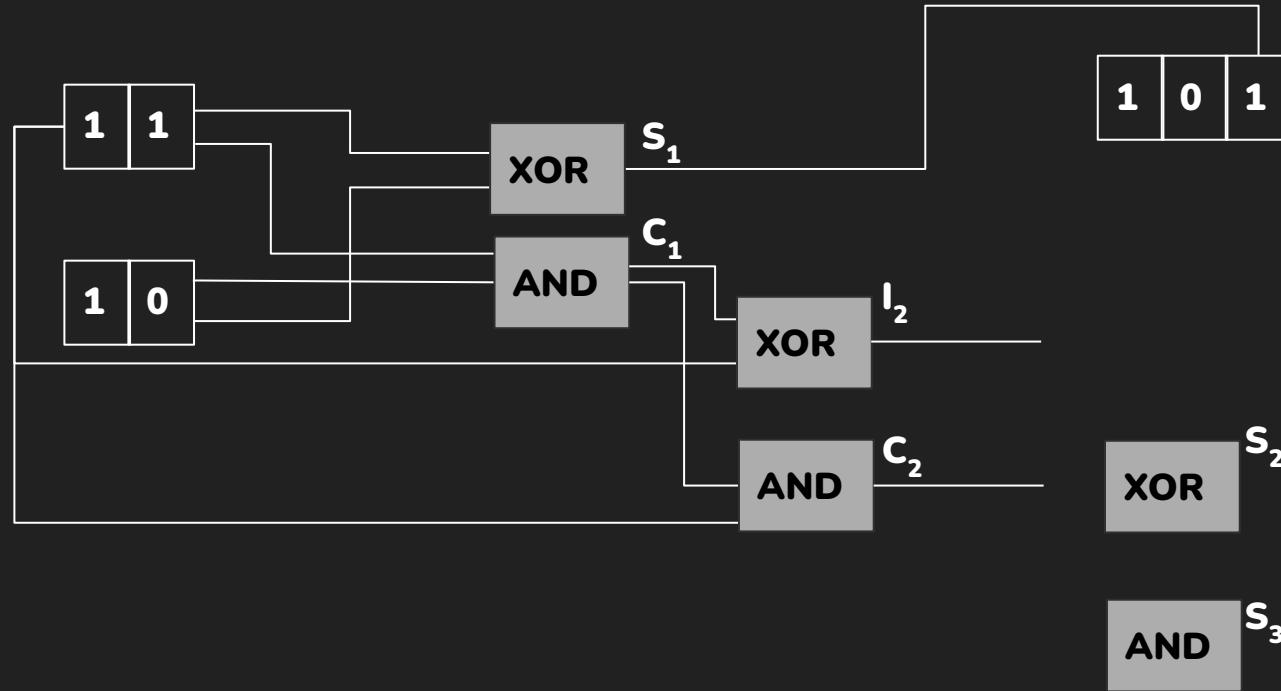
Adding 2 Numbers



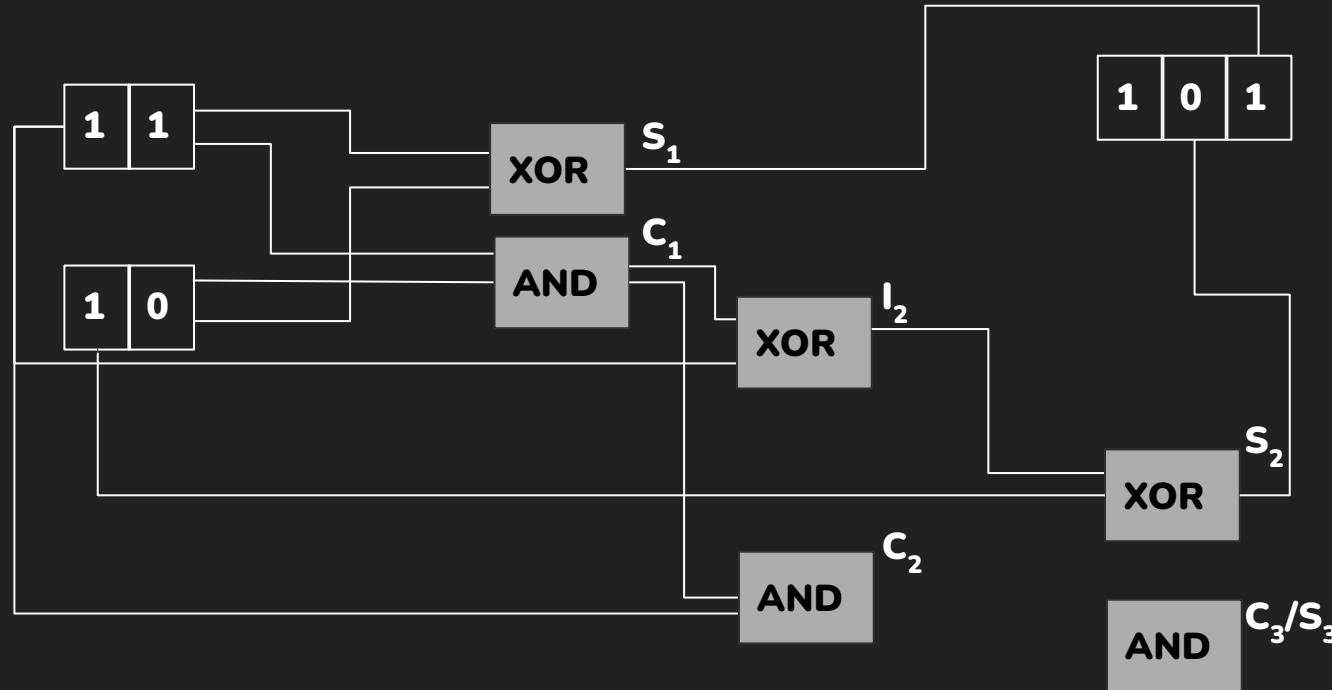
Adding 2 Numbers



Adding 2 Numbers



Adding 2 Numbers

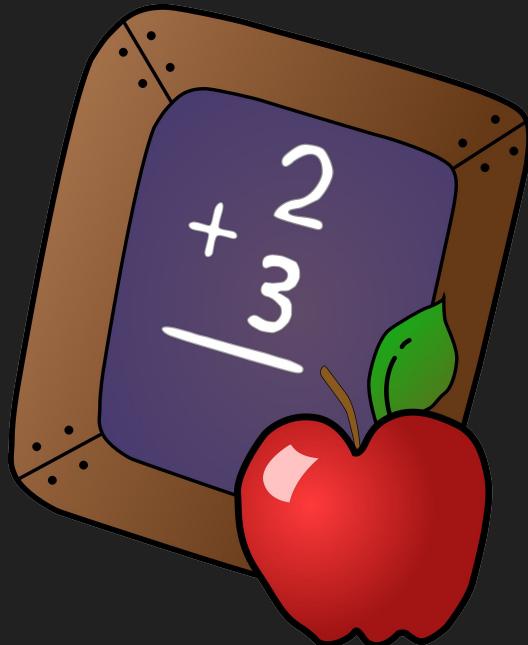


Adding 5 Numbers

```
>>> print(-5+3+(-13)+124+125124)  
125233
```

```
>>> n1 = -5  
>>> n2 = 3  
>>> n3 = -13  
>>> n4 = 124  
>>> n5 = 125124  
>>> s1 = n1 + n2  
>>> s2 = s1 + n3  
>>> s3 = s2 + n4  
>>> s4 = s3 + n5  
>>> print(s4)  
125233
```

Adding N Numbers



552

+

12

+

103

+

•

•

•

Adding N Numbers

Inputs from user: Read n, Read n numbers one by one

Output: print n and sum of n numbers

Think about the algorithm...

Hint1: add 2 numbers every time you read a new number from user

Hint2: Repeat n times

Loops

Style 1: Repeat n times (n = 20):

- 1) Let i be integer**
- 2) for i = 0 to 19 in steps of 1, repeat the following instruction:**
 - 2.1) do something**

Loops

Style 2: Repeat n times (n = 20)

- 1) Let i be integer
- 2) Set i to 0
- 3) While i is less than or equal to 19, repeat the following instructions:
 - 3.1) do something,
 - 3.2) and increment i by 1

Adding N Numbers

Algorithm :

- 1) Let n, sum_n, and x be integers, sum_n is the variable to store the sum of n numbers**
- 2) Read the value of n from user**
- 3) Set sum_n to 0**
- 4) Repeat n times, following two instructions:**
 - 4.1) Read value of x from user**
 - 4.2) add value of x to value of sum_n and store the sum in sum_n**
- 5. Print n and sum_n**

Adding N Numbers

Pseudo-code:

1) Let n, sum_n, x be integers

2) Read n

3) sum_n <- 0

4) Repeat n times:

4.1) Read x

4.2) sum_n <- sum_n + x

5. Print n, sum_n

Adding N Numbers

Pseudo-code:

1) Let n, sum_n, x be integers

2) Read n

3) sum_n <- 0

4) Repeat n times:

4.1) Read x

4.2) sum_n <- sum_n + x

5. Print n, sum_n

Python code:

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0, n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")
n = int(n)
sum_n = 0
for i in range(0,n):
    x = input("Enter x: ")
    sum_n = sum_n \
            + int(x)
print("n =", n, \
      "and sum =", sum_n)
```

- \ means instead of completing the statement in current line, we are continuing it in next line
- Space or tab does not matter in next line after using \
- Used to keep code compact (towards left)

Python uses indentation (tab or two spaces or four spaces) to denote that everything after this indentation falls under the loop or condition it is following.

**Either use tab
Or use double space
Or use four spaces**

But be consistent

You can even use single space and any combination of above options, but have to remain consistent within a condition or a loop.

We will see conditions in the next class

Adding N Numbers

```
n = input("Enter n: ")#line1
```

```
n = input("Enter n: ")
n = int(n)
sum_n = 0
for i in range(0,n):
    x = input("Enter x: ")
    sum_n = sum_n \
            + int(x)
print("n =", n, \
      "and sum =", sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")#line1  
print("line1 n type(n):",n, type(n))
```

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0,n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0,n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

```
n = input("Enter n: ")#line1  
print("line1 n type(n):",n, type(n))  
n = int(n)#line2  
print("line2 n type(n):",n, type(n))  
sum_n = 0 #line3  
print("line3 n sum_n:",n, sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0,n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

```
n = input("Enter n: ")#line1  
print("line1 n type(n):",n, type(n))  
n = int(n)#line2  
print("line2 n type(n):",n, type(n))  
sum_n = 0 #line3  
print("line3 n sum_n:",n, sum_n)  
for i in range(0,n): #line4  
    x = input("Enter x: ") #line5  
    print("line5 n sum_n x i:" \  
          ,n, sum_n, x, i)  
    sum_n = sum_n \  
           + int(x) #line6  
    print("line6 n sum_n x i:" \  
          ,n, sum_n, x, i)  
print("n =", n, \  
      "and sum =", sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")#line1
print("line1 n type(n):",n, type(n) )
n = int(n)#line2
print("line2 n type(n):",n, type(n) )
sum_n = 0 #line3
print("line3 n sum_n:",n, sum_n)
for i in range(0,n): #line4
    x = input("Enter x: ") #line5
    print("line5 n sum_n x i:" \
          ,n, sum_n, x, i)
    sum_n = sum_n\
            + int(x) #line6
    print("line6 n sum_n x i:" \
          ,n, sum_n, x, i)
print("n =", n, \
      "and sum =", sum_n)
```

Enter n:

Adding N Numbers

```
n = input("Enter n: ")#line1
print("line1 n type(n):",n, type(n) )
n = int(n)#line2
print("line2 n type(n):",n, type(n) )
sum_n = 0 #line3
print("line3 n sum_n:",n, sum_n)
for i in range(0,n): #line4
    x = input("Enter x: ") #line5
    print("line5 n sum_n x i:" \
          ,n, sum_n, x, i)
    sum_n = sum_n\
            + int(x) #line6
    print("line6 n sum_n x i:" \
          ,n, sum_n, x, i)
print("n =", n, \
      "and sum =", sum_n)
```

```
Enter n: 3
line1 n type(n): 3 <class 'str'>
line2 n type(n): 3 <class 'int'>
line3 n sum_n: 3 0
Enter x:
```

Adding N Numbers

```
n = input("Enter n: ")#line1
print("line1 n type(n):",n, type(n))
n = int(n)#line2
print("line2 n type(n):",n, type(n))
sum_n = 0 #line3
print("line3 n sum_n:",n, sum_n)
for i in range(0,n): #line4
    x = input("Enter x: ") #line5
    print("line5 n sum_n x i:" \
          ,n, sum_n, x, i)
    sum_n = sum_n\
           + int(x) #line6
    print("line6 n sum_n x i:" \
          ,n, sum_n, x, i)
print("n =", n, \
      "and sum =", sum_n)
```

```
Enter n: 3
line1 n type(n): 3 <class 'str'>
line2 n type(n): 3 <class 'int'>
line3 n sum_n: 3 0
Enter x: 1
line5 n sum_n x i: 3 0 1 0
line6 n sum_n x i: 3 1 1 0
Enter x:
```

Adding N Numbers

```
n = input("Enter n: ")#line1
print("line1 n type(n):",n, type(n))
n = int(n)#line2
print("line2 n type(n):",n, type(n))
sum_n = 0 #line3
print("line3 n sum_n:",n, sum_n)
for i in range(0,n): #line4
    x = input("Enter x: ") #line5
    print("line5 n sum_n x i:" \
          ,n, sum_n, x, i)
    sum_n = sum_n\
           + int(x) #line6
    print("line6 n sum_n x i:" \
          ,n, sum_n, x, i)
print("n =", n, \
      "and sum =", sum_n)
```

```
Enter n: 3
line1 n type(n): 3 <class 'str'>
line2 n type(n): 3 <class 'int'>
line3 n sum_n: 3 0
Enter x: 1
line5 n sum_n x i: 3 0 1 0
line6 n sum_n x i: 3 1 1 0
Enter x: 2
line5 n sum_n x i: 3 1 2 1
line6 n sum_n x i: 3 3 2 1
Enter x:
```

Adding N Numbers

```
n = input("Enter n: ")#line1
print("line1 n type(n):",n, type(n))
n = int(n)#line2
print("line2 n type(n):",n, type(n))
sum_n = 0 #line3
print("line3 n sum_n:",n, sum_n)
for i in range(0,n): #line4
    x = input("Enter x: ") #line5
    print("line5 n sum_n x i:" \
          ,n, sum_n, x, i)
    sum_n = sum_n\
           + int(x) #line6
    print("line6 n sum_n x i:" \
          ,n, sum_n, x, i)
print("n =", n, \
      "and sum =", sum_n)
```

```
Enter n: 3
line1 n type(n): 3 <class 'str'>
line2 n type(n): 3 <class 'int'>
line3 n sum_n: 3 0
Enter x: 1
line5 n sum_n x i: 3 0 1 0
line6 n sum_n x i: 3 1 1 0
Enter x: 2
line5 n sum_n x i: 3 1 2 1
line6 n sum_n x i: 3 3 2 1
Enter x: 3
line5 n sum_n x i: 3 3 3 2
line6 n sum_n x i: 3 6 3 2
n = 3 and sum = 6
```

Adding N Numbers

```
n = input("Enter n: ")
n = int(n)
sum_n = 0
for i in range(0,n):
    x = input("Enter x: ")
    sum_n = sum_n \
            + int(x)
print("n =", n, \
      "and sum = ", sum_n)
```

```
Enter n: 3
line1 n type(n): 3 <class 'str'>
line2 n type(n): 3 <class 'int'>
line3 n sum_n: 3 0
Enter x: 1
line5 n sum_n x i: 3 0 1 0
line6 n sum_n x i: 3 1 1 0
Enter x: 2
line5 n sum_n x i: 3 1 2 1
line6 n sum_n x i: 3 3 2 1
Enter x: 3
line5 n sum_n x i: 3 3 3 2
line6 n sum_n x i: 3 6 3 2
n = 3 and sum = 6
```

Adding N Numbers

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0,n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
i = 0  
while(i<n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
    i = i+1 #same as i += 1  
print("n =", n, \  
      "and sum =", sum_n)
```

Adding N Numbers

```
n = input("Enter n: ")
n = int(n)
sum_n = 0
for i in range(0,n):
    x = input("Enter x: ")
    sum_n = sum_n \
            + int(x)
print("n =", n, \
      "and sum = ", sum_n)
```

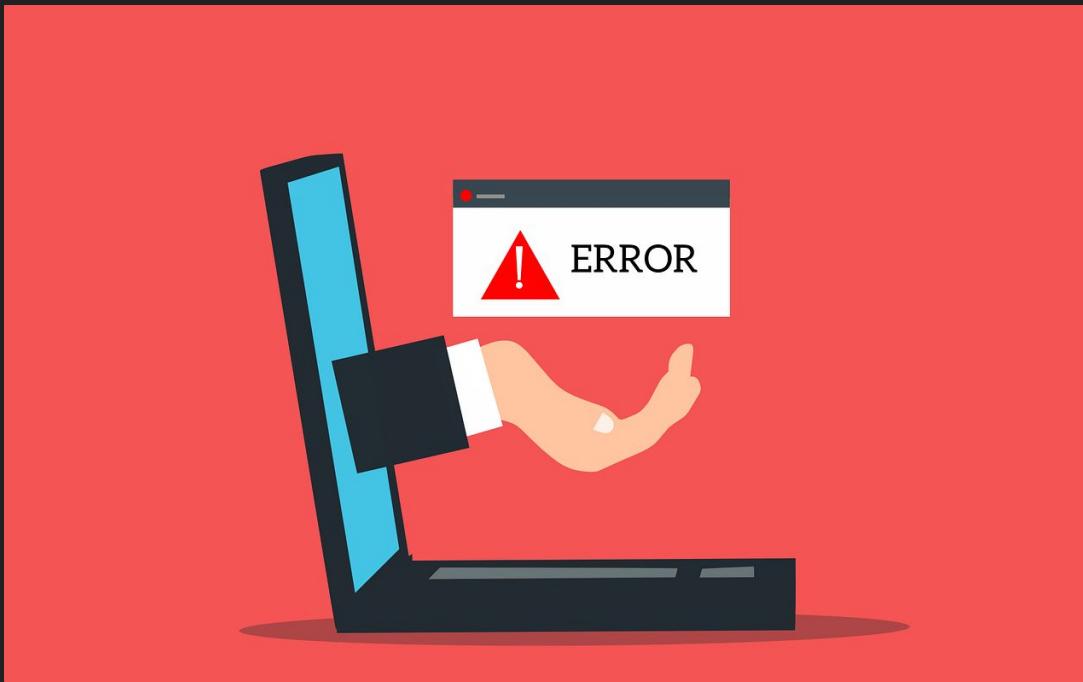
Adding N Numbers

Reducing Statements

```
n = input("Enter n: ")  
n = int(n)  
sum_n = 0  
for i in range(0,n):  
    x = input("Enter x: ")  
    sum_n = sum_n \  
           + int(x)  
print("n =", n, \  
      "and sum =", sum_n)
```

```
n = int(input("Enter n: "))  
sum_n = 0  
for i in range(0,n): sum_n = sum_n + int(input("Enter x: "))  
print("n =", n, "and sum =", sum_n)
```

Handling Errors due to User Inputs



Handling Errors due to User Inputs

```
>>> myInt = input("give me an integer: ")
give me an integer: "ha ha ha"
>>> myInt = int(myInt)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    myInt = int(myInt)
ValueError: invalid literal for int() with base 10: '"ha ha ha"'
```

Conditional Statements



Conditional Statements

Style1

- if a condition is true, then run the following instructions:
 - instruction 1
 - instruction 2

Conditional Statements

Style2

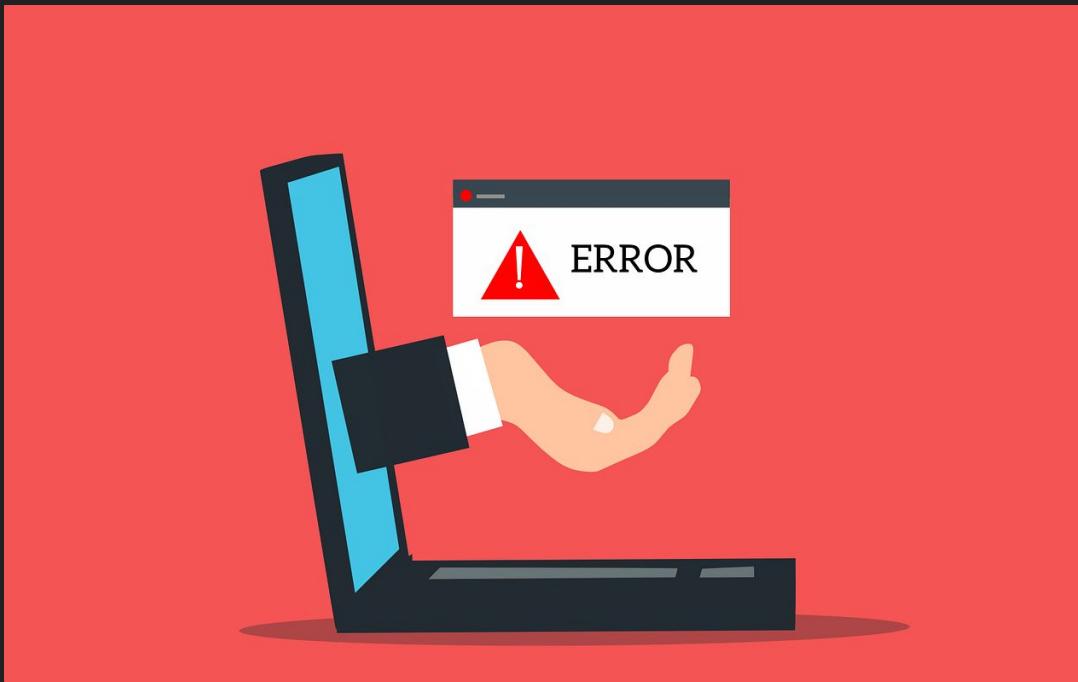
- if a condition is true, then run the following instructions:
 - instruction 1
 - instruction 2 ...
- else, run the following instructions:
 - else_instruction 1
 - else_instruction 2 ...

Conditional Statements

Style3

- if a condition is true, then run the following instructions:
 - instruction 1
 - instruction 2 ...
- else if second condition is true, run the following instructions:
 - else_if_1_instruction 1
 - else_if_1_instruction 2 ...
- else if third condition is true, run...
- .
- .
- else, run the following instructions:
 - else_instruction 1
 - else_instruction 2 ...

Handling Errors due to User Inputs



Handling Errors due to User Inputs

Algorithm:

- Prompt user to type a string with an integer value
- Until any of the character of the user's string is not a digit, prompt the user again and again to type an integer
- If all the characters of user's string are digits, print "thank you for giving me: " and the value of integer the user has typed

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
Don't think I am a fool, integer please: "oh no"
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
Don't think I am a fool, integer please: "oh no"
Don't think I am a fool, integer please: "how do you know?"
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
Don't think I am a fool, integer please: "oh no"
Don't think I am a fool, integer please: "how do you know?"
Don't think I am a fool, integer please: "ok"
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
Don't think I am a fool, integer please: "oh no"
Don't think I am a fool, integer please: "how do you know?"
Don't think I am a fool, integer please: "ok"
Don't think I am a fool, integer please: 34
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: "ha ha ha"
Don't think I am a fool, integer please: "oh no"
Don't think I am a fool, integer please: "how do you know?"
Don't think I am a fool, integer please: "ok"
Don't think I am a fool, integer please: 34
Thank you for giving me: 34
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

How to implement isdigit() function?

Traversing Strings with Loops

```
#printStringLettersEasy  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'
```

Traversing Strings with Loops

```
#printStringLettersEasy  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'  
for letter in flower:
```

Traversing Strings with Loops

```
#printStringLettersEasy  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'  
for letter in flower:  
    print(letter)
```

l
o
t
u
s

Handling Errors due to User Inputs

```
#forceDigitInput  
#Description: repeatedly prompts user to input a digit,  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja|
```

Handling Errors due to User Inputs

```
#forceDigitInput  
#Description: repeatedly prompts user to input a digit,  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja|
```

```
def myIsDigit(myString):
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja|
```

```
def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
    else:
        return False
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True

myInt = input("give me an integer: ")
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True

myInt = input("give me an integer: ")
while(myIsDigit(myInt) == False):
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True

myInt = input("give me an integer: ")
while(myIsDigit(myInt) == False):
    myInt = input("Don't think I am a fool, integer please: ")
```

Handling Errors due to User Inputs

```
#forceDigitInput
#Description: repeatedly prompts user to input a digit,
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja

def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True

myInt = input("give me an integer: ")
while(myIsDigit(myInt) == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
myInt = input("give me an integer: ")
while(myInt.isdigit() == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
give me an integer: 0
Thank you for giving me: 0
>>>
===== RESTART: D:/0000Mandi/IC152/int
give me an integer: -45
Don't think I am a fool, integer please:
```

Handling Errors due to User Inputs

Algorithm to handle user's input with a negative or positive integer:

- Prompt user to type a string with an integer value
- Until any of the character of the user's string is not a digit, **except the first character which can also be a positive or negative symbol**, prompt the user again and again to given an integer
- If all the characters of user's string are digits, **except the first character which can also be a positive or negative symbol**, print "thank you for giving me: " and the value of integer the user has typed

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True  
    for char in myString:  
        if char in digits:  
            pass
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True  
    for char in myString:  
        if char in digits:  
            pass  
        elif (char in "-+") and (firstIter):  
            pass  
        else:  
            raise ValueError("Input contains non-integer characters")  
    return True
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True  
    for char in myString:  
        if char in digits:  
            pass  
        elif (char in "-+") and (firstIter):  
            pass  
        else:  
            return False
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True  
    for char in myString:  
        if char in digits:  
            pass  
        elif (char in "-+") and (firstIter):  
            pass  
        else:  
            return False  
    firstIter = False
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

```
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput  
#Description: repeatedly prompts user to input an  
#until he/she provides so  
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):  
    digits = "0123456789"  
    firstIter = True  
    for char in myString:  
        if char in digits:  
            pass  
        elif (char in "-+") and (firstIter):  
            pass  
        else:  
            return False  
        firstIter = False  
    return True  
  
myInt = input("give me an integer: ")  
while(myIsInteger(myInt) == False):  
    myInt = input("Don't think I am a fool, integer please: ")  
print("Thank you for giving me:", myInt)
```

```
def myIsDigit(myString):  
    digits = "0123456789"  
    for char in myString:  
        if char in digits:  
            pass  
        else:  
            return False  
    return True
```

Complex!

Handling Errors due to User Inputs

```
give me an integer: faf
```

```
Don't think I am a fool, integer please: -23
```

```
Thank you for giving me: -23
```

Handling Errors due to User Inputs

```
#forceIntegerInput second way
#Description: repeatedly prompts user to input an
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):
    digits = "0123456789"
```

```
def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True
```

```
myInt = input("give me an integer: ")
while(myIsInteger(myInt) == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput second way
#Description: repeatedly prompts user to input an
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):
    digits = "0123456789"
    if myString[0] in "-+":
        myString = myString[1:]
```

```
def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True
```

```
myInt = input("give me an integer: ")
while(myIsInteger(myInt) == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

Handling Errors due to User Inputs

```
#forceIntegerInput second way
#Description: repeatedly prompts user to input an
#until he/she provides so
#written on 13 Nov.'22 by Rohit Saluja
```

```
def myIsInteger(myString):
    digits = "0123456789"
    if myString[0] in "-+":
        myString = myString[1:]
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True
```

```
myInt = input("give me an integer: ")
while(myIsInteger(myInt) == False):
    myInt = input("Don't think I am a fool, integer please: ")
print("Thank you for giving me:", myInt)
```

```
def myIsDigit(myString):
    digits = "0123456789"
    for char in myString:
        if char in digits:
            pass
        else:
            return False
    return True
```

EASY!

Handling Errors due to User Inputs

```
give me an integer: faf
```

```
Don't think I am a fool, integer please: -23
```

```
Thank you for giving me: -23
```

How to solve a problem by Computer?

- **Understand the problem**
- **Algorithm -> pseudo-code -> python code**
- **Test various inputs to check the program**
- **Debugging: fix errors in the program**

How to solve a problem by Computer?

Types of Errors

- Syntax errors:
 - E.g. `print(3+(2*9)`
 - `Print(3+(2*9))`
- Logical errors:
 - E.g. `print((-b + math.sqrt(b**2 - 4ac))/2*a)`
 - May be more difficult to fix
 - Experience...

Traversing Strings with Loops

```
flower = 'lotus'    line = 'lotus is      a\tbeautiful\nflower.'
```



l	lotus
o	is
t	a
u	beautiful
s	flower



Traversing Strings with Loops

```
#printStringLetters  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'
```

Traversing Strings with Loops

```
#printStringLetters  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'  
index = 0
```

Traversing Strings with Loops

```
#printStringLetters
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
index = 0
while index < len(flower):
```

Traversing Strings with Loops

```
#printStringLetters
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
index = 0
while index < len(flower):
    letter = flower[index]
```

Traversing Strings with Loops

```
#printStringLetters
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
index = 0
while index < len(flower):
    letter = flower[index]
    print(letter)
```

Traversing Strings with Loops

```
#printStringLetters
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
index = 0
while index < len(flower):
    letter = flower[index]
    print(letter)
    index += 1 # same as index = index + 1
```

Traversing Strings with Loops

```
#printStringLetters
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
index = 0
while index < len(flower):
    letter = flower[index]
    print(letter)
    index += 1 # same as index = index + 1
```

l
o
t
u
s

Traversing Strings with Loops

```
#printStringLettersEasy
#prints the letters of a string line by line
#written by user name on dd/mm/yyyy

flower = 'lotus'
for letter in flower:
    print(letter)
```

Traversing Strings with Loops

```
#printStringLettersEasy  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'  
for letter in flower:  
    print(letter)
```

```
#printStringLetters  
#prints the letters of a string line by line  
#written by user name on dd/mm/yyyy  
  
flower = 'lotus'  
index = 0  
while index < len(flower):  
    letter = flower[index]  
    print(letter)  
    index += 1 # same as index = index + 1
```

l
o
t
u
s

Traversing Strings with Loops

How to traverse words of a line?

```
#printStringWords
#prints the words of a string line by line
#written by user name on dd/mm/yyyy

line = 'lotus is    a\tbeautiful\nflower.'
```

Traversing Strings with Loops

How to traverse words of a line?

```
#printStringWords
#prints the words of a string line by line
#written by user name on dd/mm/yyyy

line = 'lotus is    a\tbeautiful\nflower.'
for word in line.split():
```

Traversing Strings with Loops

How to traverse words of a line?

```
#printStringWords
#prints the words of a string line by line
#written by user name on dd/mm/yyyy

line = 'lotus is    a\tbeautiful\nflower.'
for word in line.split():
    print(word)
```

```
lotus
is
a
beautiful
flower
```

String Replacements with loop

```
thought = "Anyone can learn python, python is so easy. Best way to learn  
python is using it for different tasks. We will use python to correct the text  
errors in a file in the next assignment."
```

How to replace the even instances of “python” in above string using a “for” loop?

Auto-correct contents of a text file

input.txt

```
features of Python
Python has following properties:
easy to use: python is said to be programming at the speed of thought(nice!).
simple But powerful: though python is simple , it's as powerful as other modern languages .
```



output.txt

```
Features Of Python
Python has following properties:
Easy To Use: Python is said to be programming at the speed of thought (nice!).
Simple But Powerful: Though python is simple, it's as powerful as other modern languages.
```

Reading lines from a file

input.txt

```
features of Python
Python has following properties:
easy to use: python is said to be programming at the speed of thought(nice!).
simple But powerful: though python is simple , it's as powerful as other modern languages .
```

```
#readTextFile
#program to read and print lines from a text file
#written by User Name on dd/mm/yyyy

with open('input.txt') as f:
    for line in f:
        print(line.strip())
```

Reading lines from a file

input.txt

features of Python

Python has following properties:

easy to use: python is said to be programming at the speed of thought(nice!).

simple But powerful: though python is simple , it's as powerful as other modern languages .

```
#readTextFile
#program to read and print lines from a text file
#written by User Name on dd/mm/yyyy

with open('input.txt') as f:
    for line in f:
        print(line.strip())
```

Reading lines from a file

input.txt

features of Python
Python has following properties:
easy to use: python is said to be programming at the speed of thought(nice!).
simple But powerful: though python is simple , it's as powerful as other modern languages .

```
#readTextFile
#program to read and print lines from a text file
#written by User Name on dd/mm/yyyy

with open('input.txt') as f:
    for line in f:
        print(line.strip())
```

features of Python
Python has following properties:
easy to use: python is said to be programming at the speed of tho
ught(nice!) .
simple But powerful: though python is simple , it's as powerful a
s other modern languages .

Reading lines from a file and writing to another

```
#readWriteTextFile  
#program to read lines from a text file  
#and write it to another file  
#written by User Name on dd/mm/yyyy  
  
fOutput = open('output.txt', 'w')
```

Reading lines from a file and writing to another

```
#readWriteTextFile
#program to read lines from a text file
#and write it to another file
#written by User Name on dd/mm/yyyy

fOutput = open('output.txt', 'w')
with open('input.txt') as f:
    for line in f:
        #print(line.strip())
```

Reading lines from a file and writing to another

```
#readWriteTextFile
#program to read lines from a text file
#and write it to another file
#written by User Name on dd/mm/yyyy

fOutput = open('output.txt', 'w')
with open('input.txt') as f:
    for line in f:
        #print(line.strip())
        fOutput.write(line.strip().replace(" .", ".") + "\n")
```

Reading lines from a file and writing to another

```
#readWriteTextFile
#program to read lines from a text file
#and write it to another file
#written by User Name on dd/mm/yyyy

fOutput = open('output.txt', 'w')
with open('input.txt') as f:
    for line in f:
        #print(line.strip())
        fOutput.write(line.strip().replace(" .", ".") + "\n")
fOutput.close()
```

Reading lines from a file and writing to another

```
#readWriteTxtFile
#program to read lines from a text file
#and write it to another file
#written by User Name on dd/mm/yyyy

fOutput = open('output.txt', 'w')
with open('input.txt') as f:
    for line in f:
        #print(line.strip())
        fOutput.write(line.strip().replace(" .", ".") + "\n")
fOutput.close()
```

output.txt

features of Python

Python has following properties:

easy to use: python is said to be programming at the speed of thought(nice!).

simple But powerful: though python is simple , it's as powerful as other modern languages.



Auto-correct contents of a text file

input.txt

```
features of Python
Python has following properties:
easy to use: python is said to be programming at the speed of thought(nice!).
simple But powerful: though python is simple , it's as powerful as other modern languages .
```



output.txt

```
Features Of Python
Python has following properties:
Easy To Use: Python is said to be programming at the speed of thought (nice!).
Simple But Powerful: Though python is simple, it's as powerful as other modern languages.
```

Memory Allocation

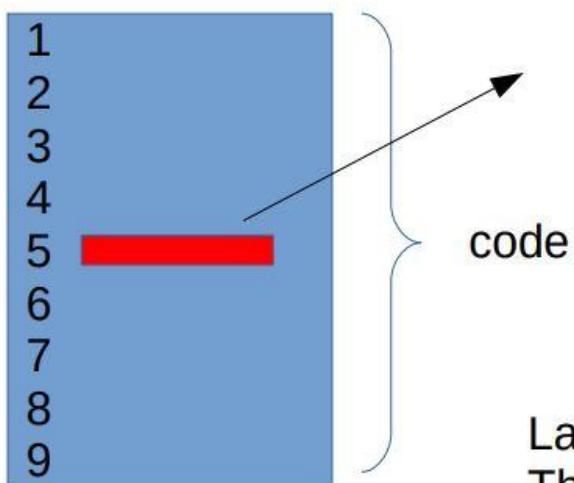
- $a, b = 1, 2$
- $c = a+b$
- $d = c * b$
- Python automatically allocates memory for a variable when it is first used
- Not in C, C++, Java. Every variable must be declared before using it.
- Memory is freed when the variable is no longer needed: Garbage Collection
- Programmer (almost) never needs to know the address of a variable

ADDRESS	DATA	VARIABLE
1028	?	
1032	1	a
1036	2	b
1040	3	c
1044	6	d
1048	?	

Python is an interpreted language
Execution happens line by line



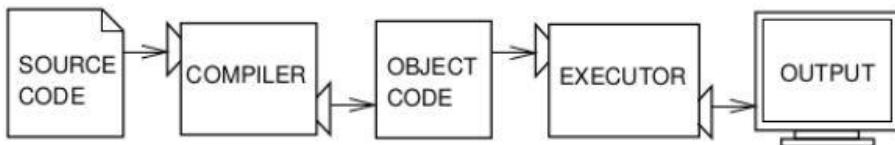
Pic from Downey's book



Syntax error in line 5

Interpreter executes till line 4 and then gives an error

Languages like C and C++ are *compiled*
The C compiler converts the program into an *executable*



Pic from Downey's book

Python: Interpreted Language

```
print('statement 1')
print('statement 2')
print('statement 3')
print('statement 4')
Print('statement 5: incorrect syntax,\n      Print used instead of print')
print('statement 6')
print('statement 7')
```

```
statement 1
statement 2
statement 3
statement 4
Traceback (most recent call last):
  File "D:/0000Mandi/assignment2/ass2.py", line 6, in <module>
    Print('statement 5: incorrect syntax,\nNameError: name 'Print' is not defined
```

Operator Precedence

Operators	Associativity
(), [], { }	left to right

Operator Precedence

Operators	Associativity
(), [], { }	left to right
func(), array[]	left to right

Operator Precedence

Operators	Associativity
(), [], { }	left to right
func(), array[]	left to right
**	<u>right to left</u>
/, //, %, *	left to right
+, -	left to right

Operator Precedence

Operators	Associativity
(), [], { }	left to right
func(), array[]	left to right
**	<u>right to left</u>
/, //, %, *	left to right
+, -	left to right
<, <=, >, >=, !=, ==	left to right

Operator Precedence

Operators	Associativity
(), [], { }	left to right
func(), array[]	left to right
**	<u>right to left</u>
/, //, %, *	left to right
+, -	left to right
<, <=, >, >=, !=, ==	left to right
not	left to right
and	left to right
Or	left to right

Will this code give any error?

```
bool1 = True  
bool2 = False  
if (bool1 == True) or (bool3):  
    print('hi')
```

LISTS

1

2

3

4

5

LISTS

- Big Data Problems
 - Thousand students in University
 - Millions of accounts in Bank
- Cannot we use one variable for each!

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))
```

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))  
average = 0
```

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))  
average = 0  
for i in range(n):
```

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))  
average = 0  
for i in range(n):  
    average += int(input()) # same as ?
```

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))  
average = 0  
for i in range(n):  
    average += int(input()) # same as ?  
average = average/n
```

LISTS

- Sometimes we can manage!

```
#averageN  
#finds average of n numbers provided by user  
#written by user name on dd/mm/yyyy  
  
n = int(input('Enter number of terms: '))  
average = 0  
for i in range(n):  
    average += int(input()) # same as ?  
average = average/n  
print('average is:', average)
```

LISTS

- Arrays
 - Variable with fixed number (say N) of entries
 - data structure containing a group of elements
 - Elements must be of same type in other languages
 - In python, they can be of any type
 - data elements stored at contiguous memory locations
- Arrays are called lists in Python

Creating and Modifying Lists

- Lists are Mutable
- In Contrast to Strings

```
>>> myList = ['1', 2, "three", 5.0 , 4]  
>>> print(myList, type(myList))  
['1', 2, 'three', 5.0, 4] <class 'list'>
```

Creating and Modifying Lists

- Lists are Mutable
- In Contrast to Strings

```
>>> myList = ['1', 2, "three", 5.0 , 4]
>>> print(myList, type(myList))
['1', 2, 'three', 5.0, 4] <class 'list'>
>>> myList[2] = "two"
```

Creating and Modifying Lists

- Lists are Mutable
- In Contrast to Strings

```
>>> myList = ['1', 2, "three", 5.0 , 4]
>>> print(myList, type(myList))
['1', 2, 'three', 5.0, 4] <class 'list'>
>>> myList[2] = "two"
>>> print(myList, type(myList))
```

Creating and Modifying Lists

- Lists are Mutable
- In Contrast to Strings

```
>>> myList = ['1', 2, "three", 5.0 , 4]
>>> print(myList, type(myList))
['1', 2, 'three', 5.0, 4] <class 'list'>
>>> myList[2] = "two"
>>> print(myList, type(myList))
['1', 2, 'two', 5.0, 4] <class 'list'>
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
>>> print(myNumbers.count(2))
3
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
>>> print(myNumbers.count(2))
3
>>> myNumbers.insert(3,15)
>>> print(myNumbers)
[1, 3, 2, 15, 6, 2, 3, 5, 2, 5]
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
>>> print(myNumbers.count(2))
3
>>> myNumbers.insert(3,15)
>>> print(myNumbers)
[1, 3, 2, 15, 6, 2, 3, 5, 2, 5]
>>> myNumbers.reverse()
```

Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
>>> print(myNumbers.count(2))
3
>>> myNumbers.insert(3,15)
>>> print(myNumbers)
[1, 3, 2, 15, 6, 2, 3, 5, 2, 5]
>>> myNumbers.reverse()
>>> print(myNumbers)
[5, 2, 5, 3, 2, 6, 15, 2, 3, 1]
```

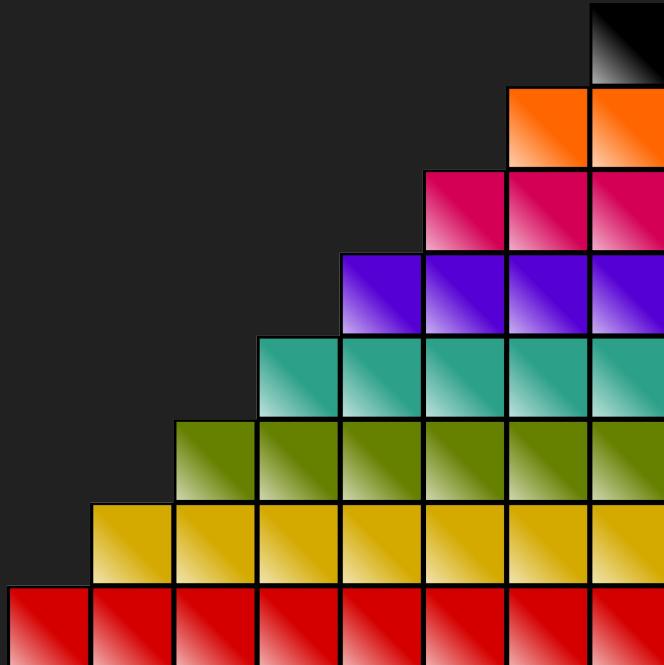
Modifying Lists

- Lists are Objects like Strings, so it has methods
- However, methods modify the list itself unlike strings

```
>>> myNumbers = [1 , 3 , 2, 6, 2 , 3, 5 , 2]
>>> myNumbers.append(5)
>>> print(myNumbers)
[1, 3, 2, 6, 2, 3, 5, 2, 5]
>>> print(myNumbers.count(2))
3
>>> myNumbers.insert(3,15)
>>> print(myNumbers)
[1, 3, 2, 15, 6, 2, 3, 5, 2, 5]
>>> myNumbers.reverse()
>>> print(myNumbers)
[5, 2, 5, 3, 2, 6, 15, 2, 3, 1]
>>> myNumbers.sort()
>>> print(myNumbers)
[1, 2, 2, 2, 3, 3, 5, 5, 6, 15]
```

Mean and Median

Mean Height
4.5 units



Median Height
[4, 5] units

Mean

- Mean or Average is ratio of sum of numbers (x_1, x_2, \dots, x_N) and total numbers (N)
- Solution of $\min_x \sum_{i=1}^N (x - x_i)^2$
- Mean of [1, 2, 4] is 7/3
- Mean of [1, 2, 4, 6] is 13/4

Median

Percentile: value below which a percentage of data falls.

Example: You are the fourth tallest person in a group of 20

80% of people are shorter than you:



That means you are at the **80th percentile**.

If your height is 1.85m then "1.85m" is the 80th percentile height in that group.

Median

- Median is the value/values in the middle of the series of values
- To get median
 - First sort the values
 - Then find the value/values in the middle
- Solution of $\min_x \sum_{i=1}^N |x - x_i|$
- Mean of elements in [1, 2, 4] is 7/3, their Median is/are
 - 2
- Mean of elements in [1, 2, 4, 6] is 13/4, their Median is/are:
 - range [2, 4]

Median (N even)

Solution of $\min_x \sum_{i=1}^N |x - x_i|$

Proof (if N is even):

- Let's assume x_1 to x_N are sorted.
- Take two terms from both ends, i.e., first and last term, i.e.:
- $s_1 = |x - x_1| + |x - x_N|$
- When will s_1 be minimum?
- $x < x_1$:
- So, clearly: $s_1 > x_N - x_1$
- $x > x_N$:
- Again, $s_1 > x_N - x_1$
- $x \in [x_1, x_N]$:
- $s_1 = x_N - x_1$



Median (N even)

x_1	x_2	x_3	x_4	x_5	x_6
-------	-------	-------	-------	-------	-------

Solution of $\min_x \sum_{i=1}^N |x - x_i|$

Proof (if N is even):

- Let's assume x_1 to x_N are sorted.
- Take sum of two terms from both ends, i.e., first and last term, i.e.:
- $s_1 = |x - x_1| + |x - x_N|$
- So, s_1 is min for $x \in [x_1, x_N]$
- Similarly, $s_2 = |x - x_2| + |x - x_{N-1}|$ will be minimum at $x \in [x_2, x_{N-1}]$
- .
- .
- $s_{N/2} = |x - x_{N/2}| + |x - x_{(N/2) + 1}|$ will be minimum at $x \in [x_{N/2}, x_{(N/2) + 1}]$
- All terms will be minimum for $x \in [x_{N/2}, x_{(N/2) + 1}]$!

Median (N odd)

x_1	x_2	x_3	x_4	x_5	x_6	x_7
-------	-------	-------	-------	-------	-------	-------

Solution of $\min_x \sum_{i=1}^N |x - x_i|$

Proof (if N is odd):

- Let's assume x_1 to x_N are sorted.
- Take sum of two terms from both ends, i.e., first and last term, i.e.:
- $s_1 = |x - x_1| + |x - x_N|$
- So, s_1 is min for $x \in [x_1, x_N]$
- Similarly, $s_2 = |x - x_2| + |x - x_{N-1}|$ will be minimum at $x \in [x_2, x_{N-1}]$
- .
- .
- $s_{(N-1)/2} = |x - x_{(N-1)/2}| + |x - x_{(N-1)/2 + 2}|$ will be minimum at $x \in [x_{(N-1)/2}, x_{(N-1)/2+2}]$
- Instead of $s_{(N+1)/2}$, we will be left with $|x - x_{(N+1)/2}|$, All min at $x = x_{(N+1)/2}$!

Median Vs Mean

Median has some advantages w.r.t. Mean.

- Who is better?
 - A team performing best consistently,
 - Or a team who performs bad all the times but outperform by large margin in just one game?
- Consider the array: [1, 1, 2, 2, 2]
- What will be its mean and median?
- Now, let's assume that there is a noise/mistake in the data:-
- [1, 1, 2, 2, 2, 100]
- 100 is added in the end by mistake
- Now, what will be mean and median?
- Median remains the same, i.e. it is less sensitive to the outliers (100 in our case).



Code to Calculate Median

```
#median
#Finds the median of user's inputs
#Written by User Name on dd/mm/yyyy

n = int(input("type total numbers you plan to input: "))
myList = []
for i in range(n):
    myList.append(int(input()))
myList.sort()
print('Debug: sorted myList:', str(myList))
if ~ (n&1): # n is even then giving average of infinite meadian
    median = (myList[int(n/2)] + myList[int(n/2)-1])/2
else: #n is odd, then giving the unique median
    median = myList[int(n/2)]
print('the median is:', median)
```

List Applications

- Representing Polynomials
- Leaderboard - Sorting Numbers
- Finding maximum continuous positive sum
- Concept of 2D Arrays lead to
- Concept of matrices, images etc.

1

2

3

4

5

Something Strange about Lists!

```
>>> l1 = [1, 2 , 3]
>>> l2 = l1
>>> l1[2] = 5
>>> print(l1)
[1, 2, 5]
>>> print(l2)
[1, 2, 5]
```

Searching index in List

```
>>> b = [] #empty list
```

```
>>> a = [1 ,3 ,5, 7, 9, 4, 5, "I am in list"]
```

```
>>> print(a.index(5))
```

2

```
>>> print(a[3:].index(5))
```

3

```
>>> print(a.index(5, 3)) # (value, start, end)
```

6

List Concatenation

```
>>> l1 = [1, 4, -2, 3]
```

```
>>> l2 = [1, 3, 4]
```

```
>>> print(l1+l2, l1)
```

```
[1, 4, -2, 3, 1, 3, 4] [1, 4, -2, 3]
```

```
>>> l1.extend(l2) # same as l1 = l1 + l2
```

```
>>> print(l1)
```

```
[1, 4, -2, 3, 1, 3, 4]
```

List Remove Vs Pop

```
>>> a = list(range(10)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
>>> a.insert(3,1) #[0, 1, 2, 1, 3, 4, 5, 6, 7, 8, 9]  
  
>>> a.remove(1) #[0, 2, 1, 3, 4, 5, 6, 7, 8, 9], removes value in argument  
  
>>> a.remove(10) #[0, 2, 1, 3, 4, 5, 6, 7, 8, 9], ValueError: list.remove(x): x  
not in list  
  
>>> p = a.pop() #p = 9, a = [0, 2, 1, 3, 4, 5, 6, 7, 8]  
  
>>> q = a.pop(1) #q = 2, a = [0, 1, 3, 4, 5, 6, 7, 8], removes value at index  
  
>>> q = a.pop(20) #q = 2, a = [0, 1, 3, 4, 5, 6, 7, 8], IndexError: pop index out  
of range
```

Sum And Product Rules



Sum Rule

Assign value 0 to variable x

Repeat n times, the following instructions:

 Assign $x + 1$ to x

Repeat n times, the following instructions:

 Assign $x + 1$ to x

Print the value of x



Product Rule

Assign value 0 to variable x

Repeat n times, the following instructions:

Repeat n times, the following instructions:

Assign $x + 1$ to x

Print the value of x



List of Lists

```
>>> my2DList = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

```
>>> print(my2DList [1])
```

```
[4, 5, 6]
```

```
>>> print(my2DList [1][2])
```

```
6
```

```
#program to loop into each child list of the parent list
```

```
for myList in my2DList:
```

```
    print('list', myList)
```

```
        for element in myList:
```

```
            print(element)
```

```
list [1, 2, 3]
1
2
3
```

List of Lists

```
>>> my2DList = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

```
>>> print(my2DList [1])
```

```
[4, 5, 6]
```

```
>>> print(my2DList [1][2])
```

```
6
```

```
#program to loop into each child list of the parent list
```

```
for myList in my2DList:
```

```
    print('list', myList)
```

```
        for element in myList:
```

```
            print(element)
```

```
list [1, 2, 3]
1
2
3
list [4, 5, 6]
4
5
6
```

List of Lists

```
>>> my2DList = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]  
  
>>> print(my2DList [1])  
[4, 5, 6]  
  
>>> print(my2DList [1][2])  
6  
  
#program to loop into each child list of the parent list  
for myList in my2DList:  
    print('list', myList)  
    for element in myList:  
        print(element)
```

```
list [1, 2, 3]  
1  
2  
3  
list [4, 5, 6]  
4  
5  
6  
list [7, 8, 9]  
7  
8  
9
```

Grayscale Image

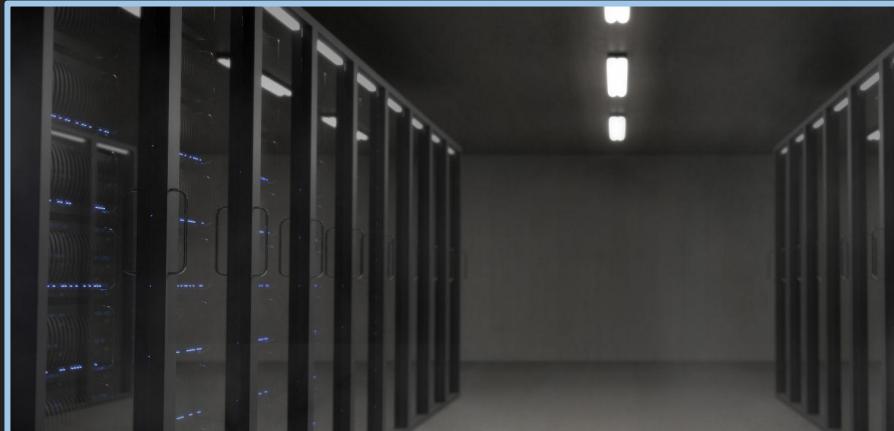


239	245	255	240	219
229	255	238	239	245
255	255	255	246	255
235	251	247	238	257
200	207	234	220	240

RGB Image



Finding Temperature of Rooms



- N (=3) sensors in each room at different locations
- readings = [[12.4, 14.19, 23.19], [10.32, 20.14, 30.53]]
- Generate alarm if average temperature > 20
- Or should we use median?
- Should we treat fire at a corner as outlier!

Finding Temperature of Rooms

```
#meanList  
#finds mean of list with three elements in two ways  
#and uses them to find average temperature of two rooms  
#written by user name on dd/mm/yyyy  
  
#first version of mean: V1  
#arguments are 3 elements of list or 3 numbers  
def meanV1(x, y, z):
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z) / 3
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z) / 3
    return mean
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z) / 3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z) / 3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2]) / 3# or sum(tempList)/3
    return mean
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z)/3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2])/3# or sum(tempList)/3
    return mean

readings = [ [12.4, 14.19, 23.19], [ 10.32, 20.14, 30.53] ]
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z)/3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2])/3# or sum(tempList)/3
    return mean

readings = [ [12.4, 14.19, 23.19], [ 10.32, 20.14, 30.53] ]
#iteratively calling first version of mean: meanV1
for i in range(len(readings)):
    print('average temperature v1 for room',i, "is:", \
        meanV1(readings[i][0], readings[i][1], readings[i][2]))
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in two ways
#and uses them to find average temperature of two rooms
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z)/3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2])/3# or sum(tempList)/3
    return mean

readings = [ [12.4, 14.19, 23.19], [ 10.32, 20.14, 30.53] ]
#iteratively calling first version of mean: meanV1
for i in range(len(readings)):
    print('average temperature v1 for room',i, "is:", \
          meanV1(readings[i][0], readings[i][1], readings[i][2]))
print()
#iteratively calling second version of mean: meanV2
for i in range(len(readings)):
    print('average temperature v2 for room',i, "is:", \
          meanV2(readings[i]))
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in
#and uses them to find average temperature
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (x + y + z)/3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2])/3# or sum(tempList)/3
    return mean

readings = [ [12.4, 14.19, 23.19], [ 10.32, 20.14, 30.53] ]
#iteratively calling first version of mean: meanV1
for i in range(len(readings)):
    print('average temperature v1 for room',i, "is:", \
          meanV1(readings[i][0], readings[i][1], readings[i][2]))
print()
#iteratively calling second version of mean: meanV2
for i in range(len(readings)):
    print('average temperature v2 for room',i, "is:", \
          meanV2(readings[i]))
```

Finding Temperature of Rooms

```
#meanList
#finds mean of list with three elements in
#and uses them to find average temperature
#written by user name on dd/mm/yyyy

#first version of mean: V1
#arguments are 3 elements of list or 3 numbers
def meanV1(x, y, z):
    #storing ratio "sum of elements: total"
    #in a variable
    mean = (x + y + z)/3
    return mean

#second version of mean: V2
#argument is a list
def meanV2(tempList):
    #storing ratio "sum of elements: total elements"
    #in a variable
    mean = (tempList[0] + tempList[1] + tempList[2])/3# or sum(tempList)/3
    return mean

readings = [ [12.4, 14.19, 23.19], [ 10.32, 20.14, 30.53] ]
#iteratively calling first version of mean: meanV1
for i in range(len(readings)):
    print('average temperature v1 for room',i, "is:", \
        meanV1(readings[i][0], readings[i][1], readings[i][2]))
print()
#iteratively calling second version of mean: meanV2
for i in range(len(readings)):
    print('average temperature v2 for room',i, "is:", \
        meanV2(readings[i]))
```

average temperature v1 for room 0 is: 16.593333333333334
average temperature v1 for room 1 is: 20.330000000000002

average temperature v2 for room 0 is: 16.593333333333334
average temperature v2 for room 1 is: 20.330000000000002

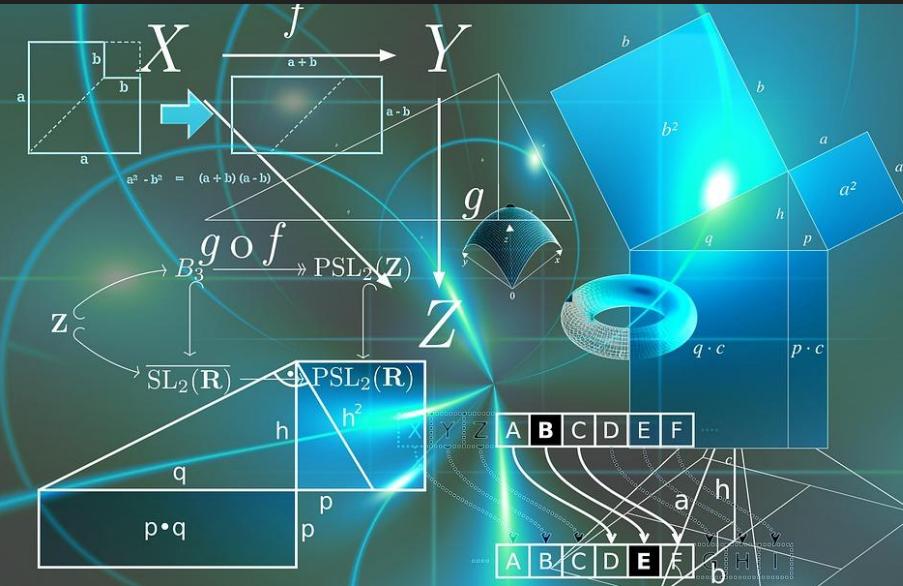
>> format(16.593333333333334, '.2f')
16.59'
>> format(float('20.3300000000002'), '.2f')
20.33'

Sorting Arrays/Lists

How to sort the following array without using sort() method?

- [5, 2, 3, 1, 9, 0]
- [0, 5, 2, 3, 1, 9]
- [0, 1, 5, 2, 3, 9]
-
-
-
-
- [0, 1, 2, 3, 5, 9]

Functions: Scope



Functions: Scope

Scope:

- region where you can unambiguously access a name
- Two types:
 - local
 - global
- Access depends on where the name is defined
- in-scope
- out-of-scope

Functions: Scope

LEGB order for name lookup

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.

Functions: Scope

Local Scope:

```
#Script to understand Local Scope:  
  
#name "base" is used for input variable  
def square(base):  
    return base ** 2
```

Functions: Scope

Local Scope:

```
#Script to understand Local Scope:  
  
#name "base" is used for input variable  
def square(base):  
    return base ** 2  
  
#name "base" is reused for input variable  
def cube(base):  
    return base ** 3
```

Functions: Scope

Local Scope:

```
#Script to understand Local Scope:  
  
#name "base" is used for input variable  
def square(base):  
    return base ** 2  
  
#name "base" is reused for input variable  
def cube(base):  
    return base ** 3  
  
inp = 8  
print(square(inp))  
print(cube(inp))  
#no confusion which base is being referred to!
```

Functions: Scope

Local Scope:

```
#Script to understand Local Scope:  
  
#name "base" is used for input variable  
def square(base):  
    return base ** 2  
  
#name "base" is reused for input variable  
def cube(base):  
    return base ** 3  
  
inp = 8  
print(square(inp))  
print(cube(inp))  
#no confusion which base is being referred to!
```

64

512

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.
- Enclosing or nonlocal scope: Only for nested functions (defined in other functions). Scope is the enclosing function.

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
```

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
    def inner():
        print('Printing var from inner function:', var)
```

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
    def inner():
        print('Printing var from inner function:', var)
    inner()
```

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
    def inner():
        print('Printing var from inner function:', var)
    inner()
    print('Printing var from outer function:', var)
```

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
    def inner():
        print('Printing var from inner function:', var)
    inner()
    print('Printing var from outer function:', var)

outer()
```

```
Printing var from inner function: 100
Printing var from outer function: 100
```

Local scope of outer() is the enclosing scope of inner()

Functions: Scope

Enclosing or
nonlocal
scope:

```
def outer():
    var = 100
    def inner():
        var = 200
        print('Printing var from inner function:', var)
    inner()
    print('Printing var from outer function:', var)

outer()
```

Printing var from inner function: 200
Printing var from outer function: 100

Local scope of outer() is the enclosing scope of inner()

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.
- Enclosing or nonlocal scope: Only for nested functions. Scope is the enclosing function.

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.
- Enclosing or nonlocal scope: Only for nested functions. Scope is the enclosing function.
- Global scope: Top-most scope.

Functions: Scope

Global scope:

```
# function f calls g, which inturn
# calls h without arguemnts, main calls f
def f():
    x = 5
    g()
    print('in f', x)
```

Functions: Scope

Global scope:

```
# function f calls g, which inturn
# calls h without arguemnts, main calls f
def f():
    x = 5
    g()
    print('in f', x)

def g():
    x = 7
    h()
```

Functions: Scope

Global scope:

```
# function f calls g, which inturn
# calls h without arguemnts, main calls f
def f():
    x = 5
    g()
    print('in f', x)

def g():
    x = 7
    h()

def h():
    print('in h', x)
```

Functions: Scope

Global scope:

```
# function f calls g, which inturn
# calls h without arguemnts, main calls f
def f():
    x = 5
    g()
    print('in f', x)

def g():
    x = 7
    h()

def h():
    print('in h', x)

x = 3
f()
print('in main', x)
```

Functions: Scope

Global scope:

```
# function f calls g, which inturn
# calls h without arguemnts, main calls f
def f():
    x = 5
    g()
    print('in f', x)

def g():
    x = 7
    h()

def h():
    print('in h', x)

x = 3
f()
print('in main', x)
```

What if h() is nested?

```
in h 3
in f 5
in main 3
```

Functions: Scope

Global scope:

```
def outer():
    #defines local scope of outer()
    #als defines enclosing scope of inner()
    def inner():
        print(number)
    inner()

number = 10
outer()
```

10

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.
- Enclosing or nonlocal scope: Only for nested functions. Scope is the enclosing function.
- Global scope: Top-most scope.

Functions: Scope

LEGB order for name lookup

- Local: Names defined inside the block. Created at function call, not at definition.
- Enclosing or nonlocal scope: Only for nested functions. Scope is the enclosing function.
- Global scope: Top-most scope.
- Built-in scope: Special scope for built-in things: keywords, exceptions etc.

Functions: Scope

Built-in scope:

```
>>> myList = [10, 2, -3]  
>>> print(sum(myList))
```

For built-in
functions,

e.g. `len()`, `sum()`

Functions: Scope

Built-in scope:

```
>>> myList = [10, 2, -3]
>>> print(sum(myList))
9
```

For built-in
functions,

e.g. `len()`, `sum()`

Functions: Scope

Built-in scope:

```
>>> myList = [10, 2, -3]
>>> print(sum(myList))
9
>>> sum = 3
```

For built-in
functions,

e.g. `len()`, `sum()`

Functions: Scope

Built-in scope:

For built-in
functions,

e.g. `len()`, `sum()`

```
>>> myList = [10, 2, -3]
>>> print(sum(myList))
9
>>> sum = 3
>>> newList = [2, 3, 4]
```

Functions: Scope

Built-in scope:

For built-in
functions,

e.g. `len()`, `sum()`

```
>>> myList = [10, 2, -3]
>>> print(sum(myList))
9
>>> sum = 3
>>> newList = [2, 3, 4]
>>> print(sum(newList))
```

Functions: Scope

Built-in scope:

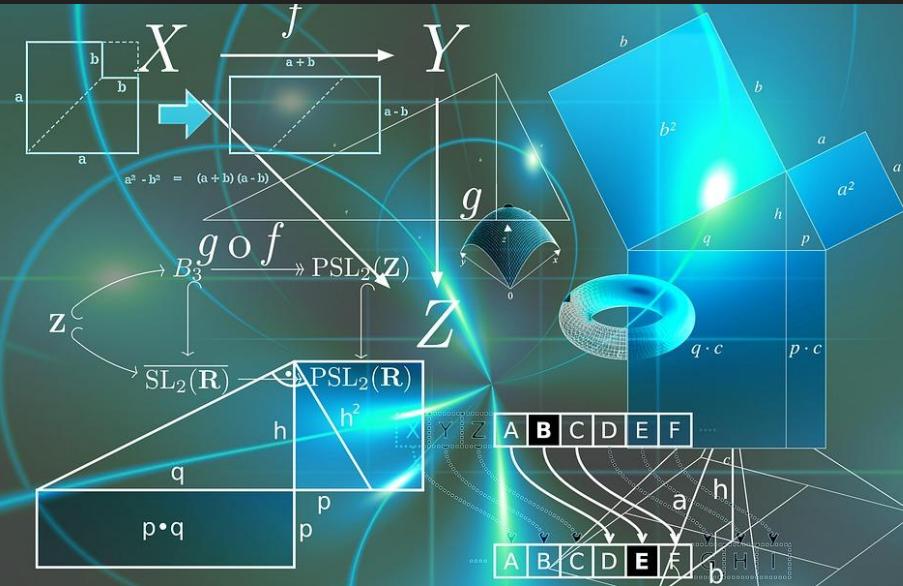
For built-in
functions,

e.g. `len()`, `sum()`

```
>>> myList = [10, 2, -3]
>>> print(sum(myList))
9
>>> sum = 3
>>> newList = [2, 3, 4]
>>> print(sum(newList))
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(sum(newList))
TypeError: 'int' object is not callable
```

Remember: do not redefine built-in names!

Functions: Arguments



Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName  
  
  
#main program:  
print(concatNameV1('Ajay', 'Gupta'))
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName  
  
  
#main program:  
print(concatNameV1('Ajay', 'Gupta'))  
print(concatNameV2('Kumari', 'Preeti'))
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName  
  
  
#main program:  
print(concatNameV1('Ajay', 'Gupta'))  
print(concatNameV2('Kumari', 'Preeti'))  
print(concatNameV3('Rahul', 'Nair'))
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName  
  
  
#main program:  
print(concatNameV1('Ajay', 'Gupta'))  
print(concatNameV2('Kumari', 'Preeti'))  
print(concatNameV3('Rahul', 'Nair'))  
print(concatNameV3('Ajay'))
```

Functions: Arguments

```
#function definitions:  
def concatNameV1(firstName, lastName):  
    return firstName + ' ' + lastName  
  
def concatNameV2(lastName, firstName):  
    return firstName + ' ' + lastName  
  
#In the following function, 2nd argument is optional  
#It has default value 'Gupta', which we can omit in its call  
def concatNameV3(firstName, lastName = 'Gupta'):  
    return firstName + ' ' + lastName  
  
#main program:  
print(concatNameV1('Ajay', 'Gupta'))  
print(concatNameV2('Kumari', 'Preeti'))  
print(concatNameV3('Rahul', 'Nair'))  
print(concatNameV3('Ajay'))
```

Ajay Gupta
Preeti Kumari
Rahul Nair
Ajay Gupta

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)  
print(a) # ['Monday', 'end']
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)  
print(a) # ['Monday', 'end']  
  
appendEnd(a)
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)  
print(a) # ['Monday', 'end']  
  
appendEnd(a)  
print(a) # ['Monday', 'end', 'end']
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)  
print(a) # ['Monday', 'end']  
  
appendEnd(a)  
print(a) # ['Monday', 'end', 'end']  
  
#Unexpected behavior for mutable objects.  
#Hence, Avoid it:-  
b = appendEnd()  
b = appendEnd()
```

Functions: Arguments

```
#function definition:  
def appendEnd(myList = []):  
    myList.append('end')  
    return myList  
  
#main program:  
a = ['Monday']  
appendEnd(a)  
print(a) # ['Monday', 'end']  
  
appendEnd(a)  
print(a) # ['Monday', 'end', 'end']  
  
#Unexpected behavior for mutable objects.  
#Hence, Avoid it:-  
b = appendEnd()  
b = appendEnd()  
print(b) # ['end', 'end']
```

Functions: Arguments

```
#function definition:  
def f(y):  
    y = 10
```

Functions: Arguments

```
#function definition:  
def f(y):  
    y = 10  
  
#main program:  
x = 5
```

Functions: Arguments

```
#function definition:  
def f(y):  
    y = 10  
  
#main program:  
x = 5  
  
print(x)  
#prints 5
```

Functions: Arguments

```
#function definition:  
def f(y):  
    y = 10  
  
#main program:  
x = 5  
  
print(x)  
#prints 5  
  
f(x)
```

Functions: Arguments

```
#function definition:  
def f(y):  
    y = 10
```

```
#main program:  
x = 5
```

```
print(x)  
#prints 5
```

```
f(x)
```

```
print(x)
```

Functions: Arguments

use id()

```
#function definition:  
def f(y):  
    y = 10  
  
#main program:  
x = 5  
  
print(x)  
#prints 5  
  
f(x)  
  
print(x)  
#prints 5
```

(main program)

x

(f)
y

1451351212

5

(main program)

x

(f)
y

1451351212

5

14513521472

10

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10
```

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10  
  
#main program:  
x = [1, 2, 3]
```

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10  
  
#main program:  
x = [1, 2, 3]  
  
print(x)  
#prints [1, 2, 3]
```

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10  
  
#main program:  
x = [1, 2, 3]  
  
print(x)  
#prints [1, 2, 3]  
  
g(x)
```

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10  
  
#main program:  
x = [1, 2, 3]  
  
print(x)  
#prints [1, 2, 3]  
  
g(x)  
  
print(x)
```

Functions: Arguments

```
#function definition:  
def g(y):  
    y[0] = 10  
  
#main program:  
x = [1, 2, 3]  
  
print(x)  
#prints [1, 2, 3]  
  
g(x)  
  
print(x)  
#prints [10, 2, 3]
```

- Contents of mutable objects can be changed
 - Immutable objects like:
 - int
 - str
 - tuple
- Cannot be changed.

Functions: Arguments

```
#function definition:-  
def doubleInput(x):  
    print("id(x) in function: ", id(x))  
    return x*2
```

Functions: Arguments

```
#function definition:-  
def doubleInput(x):  
    print("id(x) in function: ", id(x))  
    return x*2  
  
#main program:-  
x = 6  
print("id(x) in main, 1st:", id(x))
```

Functions: Arguments

```
#function definition:-  
def doubleInput(x):  
    print("id(x) in function: ", id(x))  
    return x*2  
  
#main program:-  
x = 6  
print("id(x) in main, 1st:", id(x))  
  
x = doubleInput(x)  
  
print("id(x) in main, 2nd:", id(x))
```

Functions: Arguments

```
#function definition:-  
def doubleInput(x):  
    print("id(x) in function: ", id(x))  
    return x*2  
  
#main program:-  
x = 6  
print("id(x) in main, 1st:", id(x))  
  
x = doubleInput(x)  
  
print("id(x) in main, 2nd:", id(x))  
  
''' Prints following:-  
id(x) in main, 1st: 2412663695824  
id(x) in function: 2412663695824  
id(x) in main, 2nd: 2412663696016  
'''
```

Functions: Arguments

```
#function definition:-  
def doubleInput(x):  
    print("id(x) in function: ", id(x))  
    return x*2  
  
#main program:-  
x = 6  
print("id(x) in main, 1st:", id(x))  
  
x = doubleInput(x)  
  
print("id(x) in main, 2nd:", id(x))  
  
''' Prints following:-  
id(x) in main, 1st: 2412663695824  
id(x) in function: 2412663695824  
id(x) in main, 2nd: 2412663696016  
'''
```

H/W: What if x is a list?

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp
```

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp  
  
def swapV2(a,b):  
    return b, a # returns more than one value
```

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp  
  
def swapV2(a,b):  
    return b, a # returns more than one value  
  
#main program:-  
i, j = 2, 3  
print("Original i, j:", i, j)
```

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp  
  
def swapV2(a,b):  
    return b, a # returns more than one value  
  
#main program:-  
i, j = 2, 3  
print("Original i, j:", i, j)  
  
swapV1(i,j)  
print("After swapV1:", i, j)
```

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp  
  
def swapV2(a,b):  
    return b, a # returns more than one value  
  
#main program:-  
i, j = 2, 3  
print("Original i, j:", i, j)  
  
swapV1(i,j)  
print("After swapV1:", i, j)  
  
i, j = swapV2(i,j)  
print("After swapV2:", i, j)
```

Functions: Arguments

```
#functions definition:-  
def swapV1(a,b):  
    temp = a  
    a = b  
    b = temp  
  
def swapV2(a,b):  
    return b, a # returns more than one value  
  
#main program:-  
i, j = 2, 3  
print("Original i, j:", i, j)  
  
swapV1(i,j)  
print("After swapV1:", i, j)  
  
i, j = swapV2(i,j)  
print("After swapV2:", i, j)
```

```
Original i, j: 2 3  
After swapV1: 2 3  
After swapV2: 3 2
```

Recursion

N!

Recursion

- A recursive function **calls itself**
- Factorial
 - $n! = 1 \quad n = 0$
 - $n \times (n-1)!$

Recursion

```
#function definition:-  
def factorialN(N) :
```

Recursion

```
#function definition:-  
def factorialN(N):  
    if (N==0):  
        print('Debug: N = 0')  
        return 1
```

Recursion

```
#function definition:-  
def factorialN(N):  
    if (N==0):  
        print('Debug: N = 0')  
        return 1  
    else:  
        print('Debug: N =', N)  
        return N*factorialN(N-1)
```

Recursion

```
#function definition:-  
def factorialN(N):  
    if (N==0):  
        print('Debug: N = 0')  
        return 1  
    else:  
        print('Debug: N =', N)  
        return N*factorialN(N-1)  
  
#main program:-  
x = int(input('Enter a natural number: '))
```

Recursion

```
#function definition:-  
def factorialN(N):  
    if (N==0):  
        print('Debug: N = 0')  
        return 1  
    else:  
        print('Debug: N =', N)  
        return N*factorialN(N-1)  
  
#main program:-  
x = int(input('Enter a natural number: '))  
print(factorialN(x))
```

Recursion

```
#function definition:-  
def factorialN(N):  
    if (N==0):  
        print('Debug: N = 0')  
        return 1  
    else:  
        print('Debug: N =', N)  
        return N*factorialN(N-1)
```

```
#main program:-  
x = int(input('Enter a natural number: '))  
print(factorialN(x))
```

```
Enter a natural number: 5  
Debug: N = 5  
Debug: N = 4  
Debug: N = 3  
Debug: N = 2  
Debug: N = 1  
Debug: N = 0  
120
```

Fibonacci
numbers

F_n

=

$F_{n-1} + F_{n-2}$

Recursion: Fibonacci numbers

- $n!$ has recursion based on a function $f(n-1)$
- Fibonacci numbers have recursion based on:
- $F_n = F_{n-1} + F_{n-2}$
- $F_0 = 0, F_1 = 1$
- So, $F_2 = 1, F_3 = 2, \dots$
- $\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\dots\}$

Recursion: Fibonacci numbers

```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
```

Recursion: Fibonacci numbers

```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
    if n==0:
        return 0
```

Recursion: Fibonacci numbers

```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
    if n==0:
        return 0
    if n==1:
        return 1
```

Recursion: Fibonacci numbers

```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
    if n==0:
        return 0
    if n==1:
        return 1
    return fibonacciRec(n-1) + fibonacciRec(n-2)
```

Recursion: Fibonacci numbers

```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
    if n==0:
        return 0
    if n==1:
        return 1
    return fibonacciRec(n-1) + fibonacciRec(n-2)

#main program
print(fibonacciRec(4))
```

Recursion: Fibonacci numbers

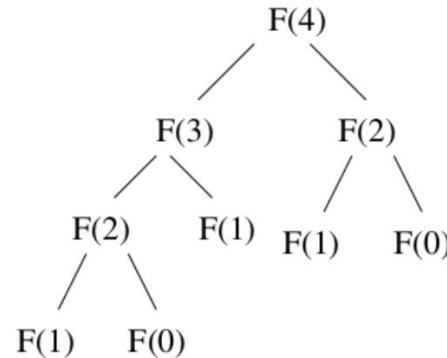
```
# fibonacciRec
# computes Fibonacci of n by recursion: exponential time

def fibonacciRec(n):
    print("in fib with value", n)
    if n==0:
        print('Debug: Base case 0')
        return 0
    if n==1:
        print('Debug: Base case 1')
        return 1
    tempSum = fibonacciRec(n-1) + fibonacciRec(n-2)
    print('Debug: Base case', n, tempSum)
    return tempSum

#main program
print(fibonacciRec(4))
```

```
in fib with value 4
in fib with value 3
in fib with value 2
in fib with value 1
Debug: Base case 1
in fib with value 0
Debug: Base case 0
Debug: Base case 2 1
in fib with value 1
Debug: Base case 1
Debug: Base case 3 2
in fib with value 2
in fib with value 1
Debug: Base case 1
in fib with value 0
Debug: Base case 0
Debug: Base case 2 1
Debug: Base case 4 3
3
```

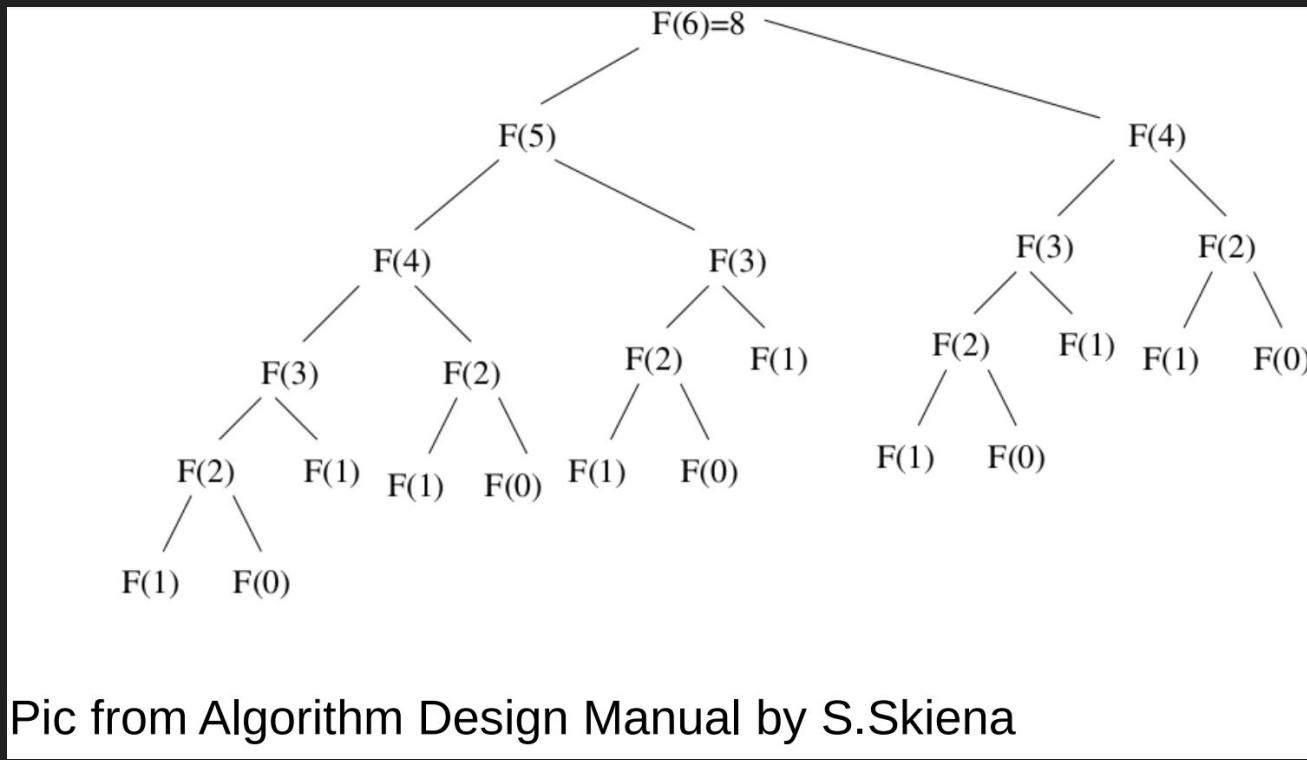
Recursion: Fibonacci numbers



Pic from Algorithm Design Manual by S.Skiena

```
in fib with value 4
in fib with value 3
in fib with value 2
in fib with value 1
Debug: Base case 1
in fib with value 0
Debug: Base case 0
Debug: Base case 2 1
in fib with value 1
Debug: Base case 1
Debug: Base case 3 2
in fib with value 2
in fib with value 1
Debug: Base case 1
in fib with value 0
Debug: Base case 0
Debug: Base case 2 1
Debug: Base case 4 3
3
```

Recursion: Fibonacci numbers



Pic from Algorithm Design Manual by S.Skiena

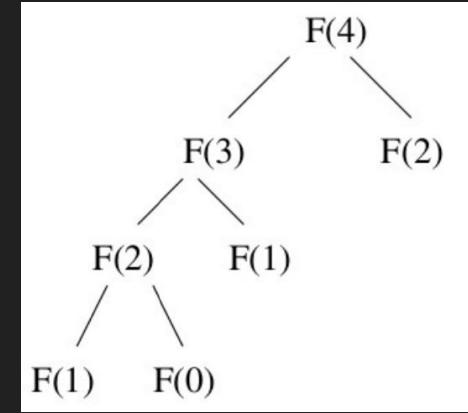
Recursion: Fibonacci numbers

- Previous program takes exponential time
- Can we do better?
- Store computed values to use later.

Recursion: Fibonacci numbers

- Store computed values to use later.
- E.g.

-1	-1	-1	-1	-1
0	1	-1	-1	-1



```
# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]
```

0	1	1	-1	-1
0	1	1	2	-1
0	1	1	2	3

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
```

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
```

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]
```

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]

# main program with 20 cache size:
nFibList = [-1]*20
```

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]

# main program with 20 cache size:
nFibList = [-1]*20
nFibList[0] = 0
nFibList[1] = 1
```

Recursion: Fibonacci numbers

```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]

# main program with 20 cache size:
nFibList = [-1]*20
nFibList[0] = 0
nFibList[1] = 1
print(fibonacciList(4))
```

Recursion: Fibonacci numbers

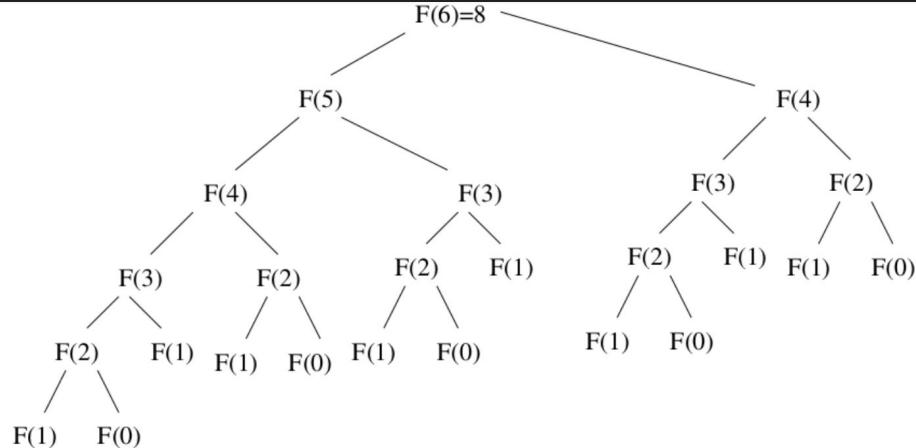
```
# fibonacciList
# computse Fibonacci of n by recursion with storage/cache/list

# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]

# main program with 20 cache size:
nFibList = [-1]*20
nFibList[0] = 0
nFibList[1] = 1
print(fibonacciList(4))
```

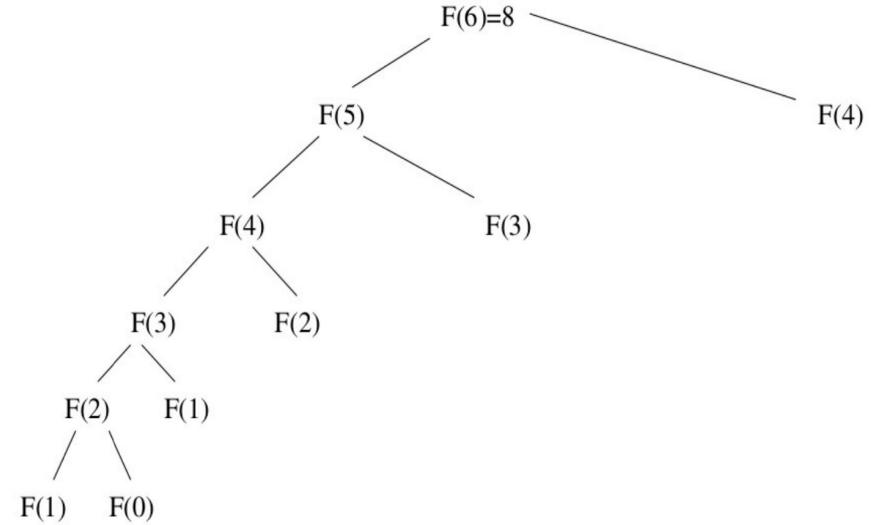
3

Recursion: Fibonacci numbers



Pic from Algorithm Design Manual by S.Skiena

WITHOUT STORAGE



WITH STORAGE

Recursion: Fibonacci numbers

```
# fibonacciRecTime
# computes time to calculate Fibonacci of n by recursion: exponential time
import time

def fibonacciRec(n):
    if n==0:
        return 0
    if n==1:
        return 1
    return fibonacciRec(n-1) + fibonacciRec(n-2)

#main program
tic = time.perf_counter()
print("fibonacci of n = 32 is:", fibonacciRec(32))
toc = time.perf_counter()
print(f'Time taken rec: {toc - tic:0.4f} seconds')
```

fibonacci of 32 is: 2178309

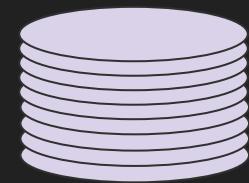
Time taken rec: 0.4562 seconds

Recursion: Fibonacci numbers

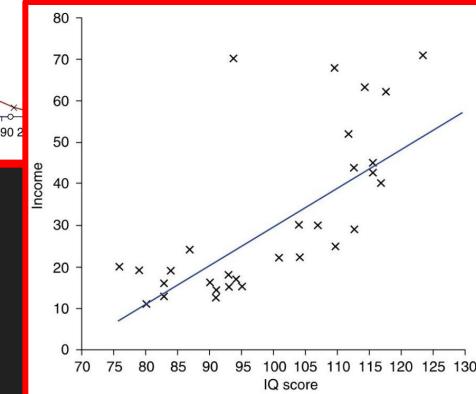
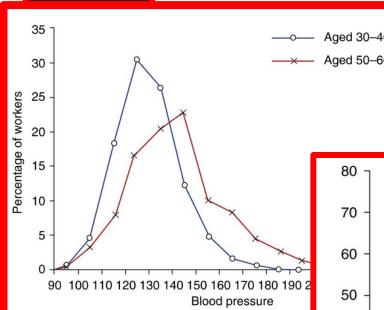
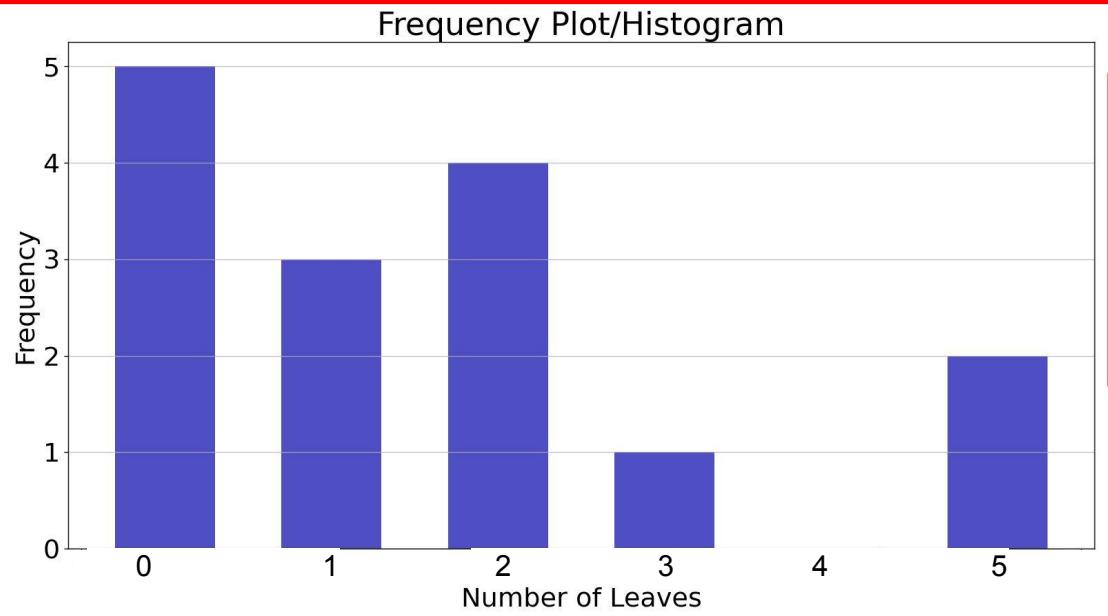
```
# fibonacciListTime
# computes time to calculate Fibonacci of n
# by recursion with storage/cache|  
  
import time  
  
# assumes nFibList[0] = 0 and nFibList[1] = 1, remaining values -1
# then finds any nFibList[n]
def fibonacciList(n):
    if nFibList[n] == -1:
        nFibList[n] = fibonacciList(n-1) + fibonacciList(n-2)
    return nFibList[n]  
  
# main program:-  
# max Fib Number
maxFib = 100  
  
# global list for caching
nFibList = [-1]*maxFib
nFibList[0] = 0
nFibList[1] = 1  
  
# input
n = 32
assert n < maxFib
tic = time.perf_counter()  
print("fibonacci of n = 32 is:". fibonacciList(n))
toc = time.perf_counter()
print(f'Time taken rec: {toc - tic:.4f} seconds')
```

fibonacci of n = 32 is: 2178309
Time taken rec: 0.0083 seconds

Describing Datasets



Value	Freq.
0	5
1	3
2	4



Describing Datasets

- Numerical findings: need to be presented concisely.
- Especially needed for large datasets.
- Features of the data include:
 - Range
 - Degree of symmetry
 - Concentrated or spread out
 - Where are they concentrated? Etc.
- Univariate or multivariate
 - One variable or multiple!

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
-------	-----------

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	3

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	3
2	4
3	1
4	

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	3
2	4
3	1
4	0

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2

Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0

Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	

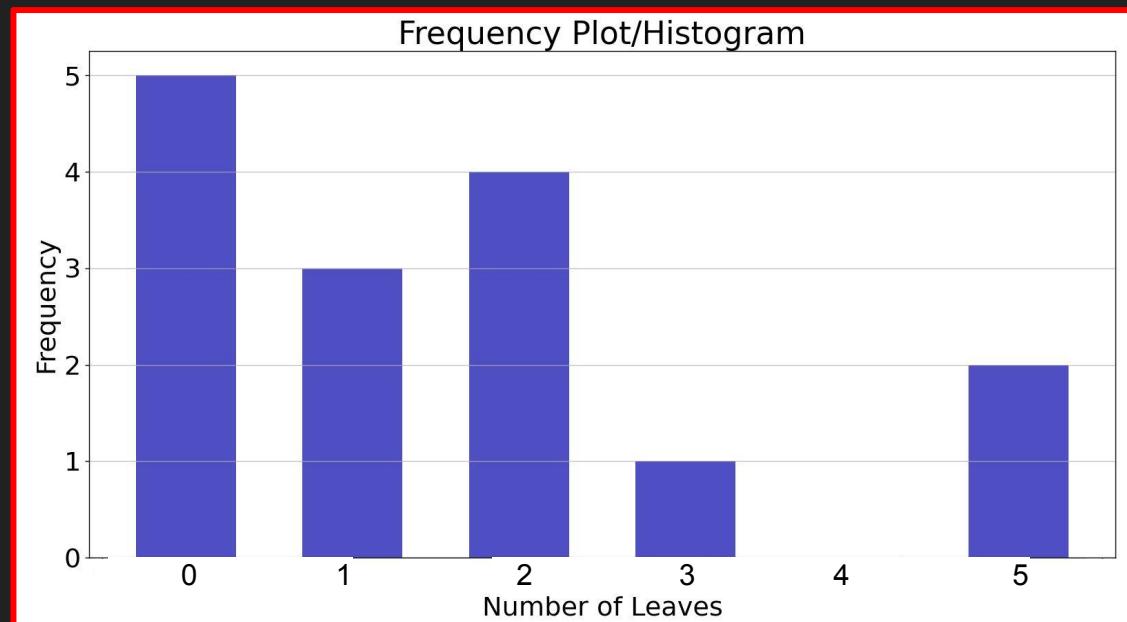
Describing Datasets

- Frequency Table for small datasets
- No. of leaves taken by 15 employees, over six weeks:
- 0, 2, 1, 2, 0, 5, 5, 0, 1, 3, 2, 0, 1, 2, 0
- How many workers had at least one day of leave?
- How many workers had leaves between 3 and 5 days?
- How many had more than 5 days of leave?

Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15

Describing Datasets: Bar Plots

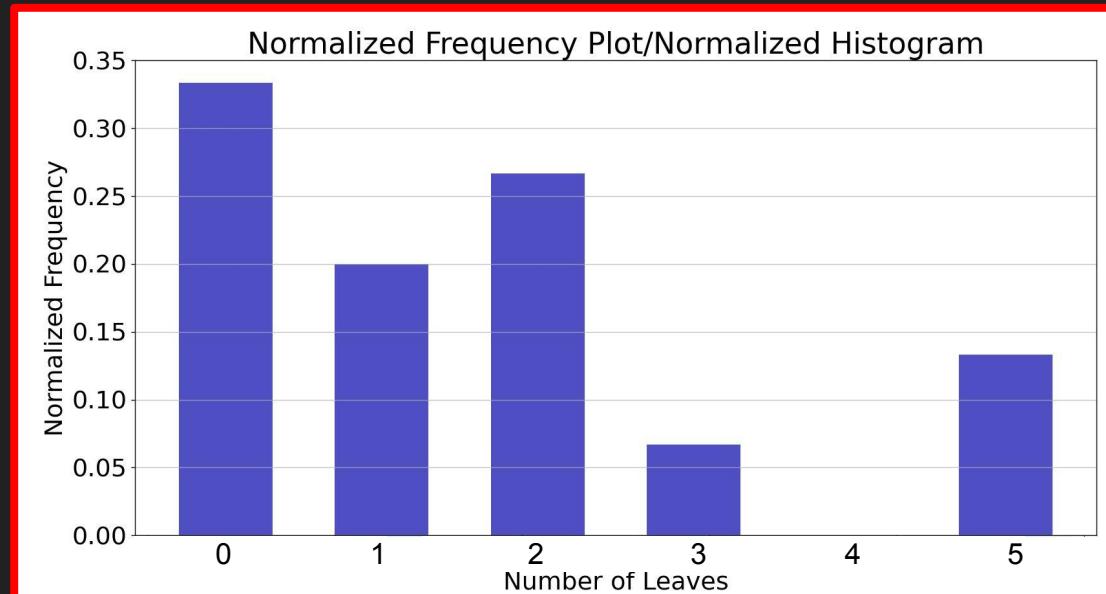
Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15



How will you construct this table and make the plot?
0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 5, 5

Describing Datasets: Bar Plots

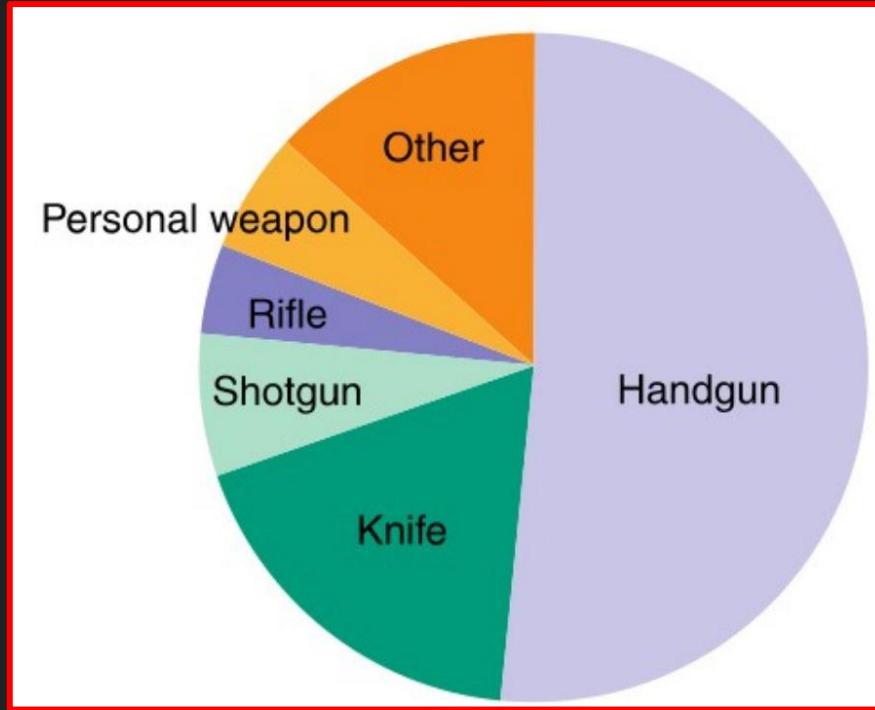
Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15



Sometimes normalized frequency is more convenient. Sum of frequency values = 1.

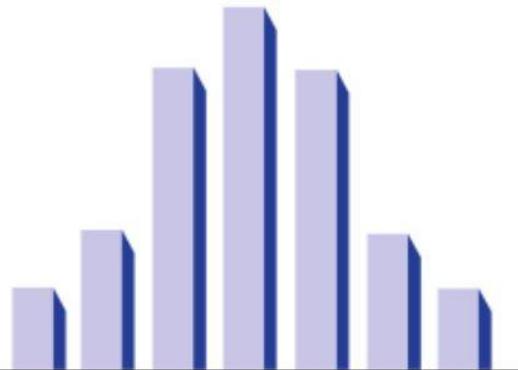
Describing Datasets: Pie Chart

Pie chart: for non-numerical data Visualization of relative frequency plot



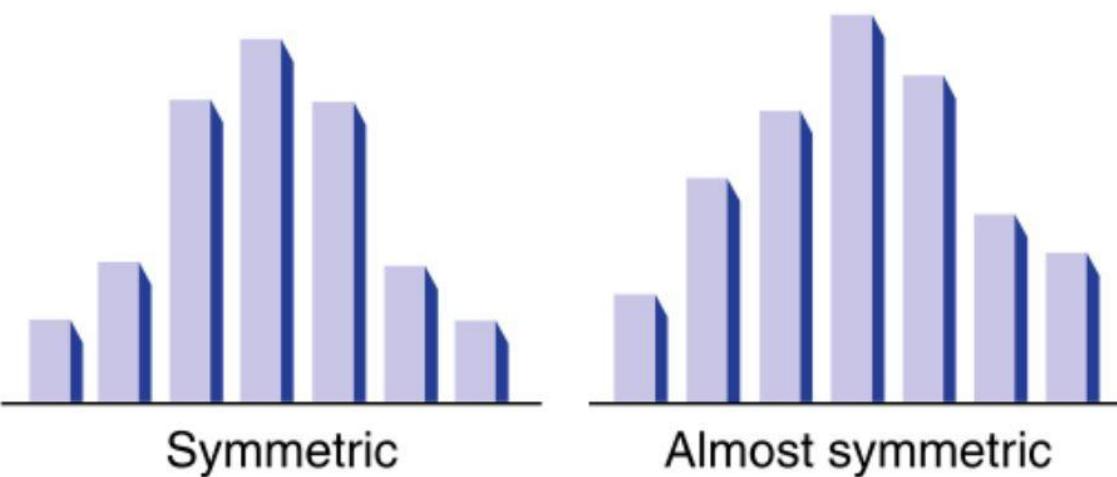
**Weapons used
in crimes.**

Bar Plot Types

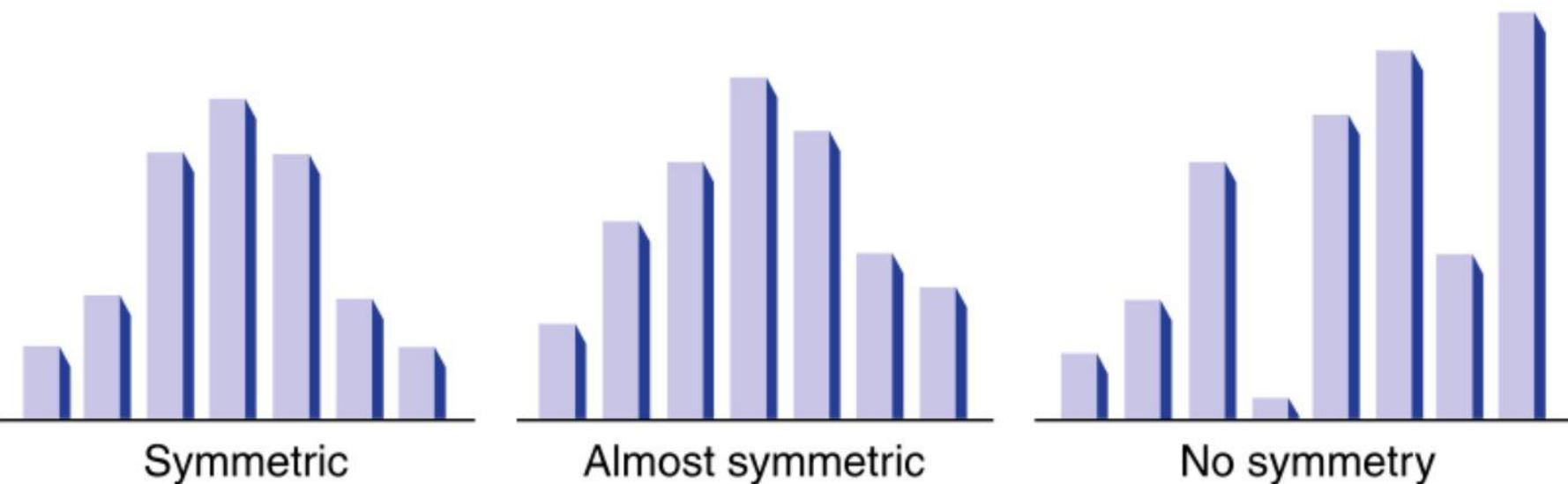


Symmetric

Bar Plot Types



Bar Plot Types

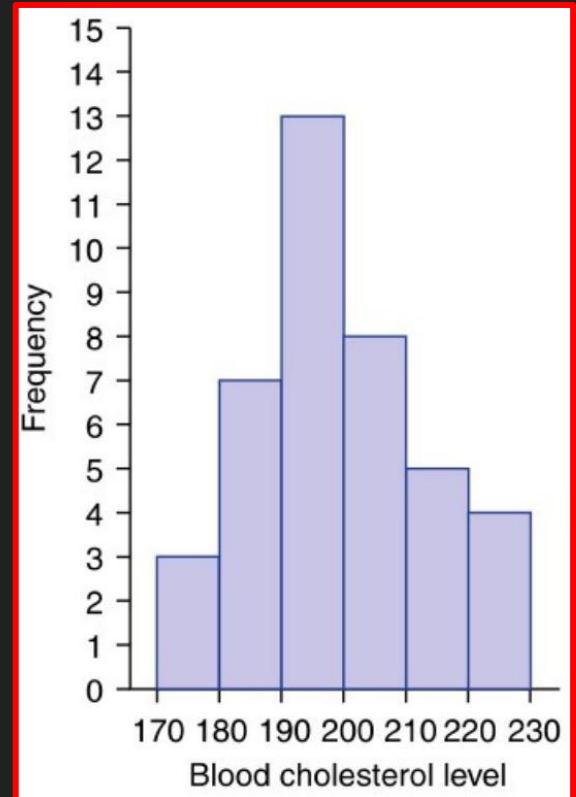


Describing Datasets: Histograms

- When unique values are numerous
- Use Bins or Class Intervals

213	174	193	196	220	183	194	200
192	200	200	199	178	183	188	193
187	181	193	205	196	211	202	213
216	206	195	191	171	194	184	191
221	212	221	204	204	191	183	227

Data: Blood cholesterol levels



Describing Datasets: Histograms

- When unique values are numerous
- Use Bins or Class Intervals

171, 174, 178, 181, 183, 183, 183, 184, 187, 188, 191,
191, 191, 191, 191, 192, 193, 193, 193, 194, 194, 194, 195
196, 196, 199, 200, 200, 200, 202, 204, 204, 205, 206,
206, 211, 212, 213, 213, 216, 220, 221, 221, 227

Sorted Blood cholesterol levels

Class interval contains left-end, but not right-end

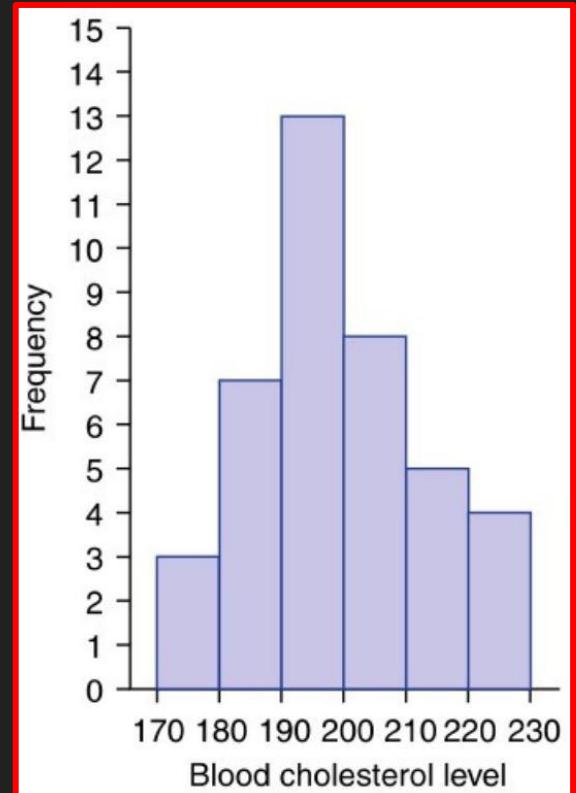
Class intervals	Frequency	Relative frequency
170–180	3	$\frac{3}{40} = 0.075$
180–190	7	$\frac{7}{40} = 0.175$
190–200	13	$\frac{13}{40} = 0.325$
200–210	8	$\frac{8}{40} = 0.20$
210–220	5	$\frac{5}{40} = 0.125$
220–230	4	$\frac{4}{40} = 0.10$

Describing Datasets: Histograms

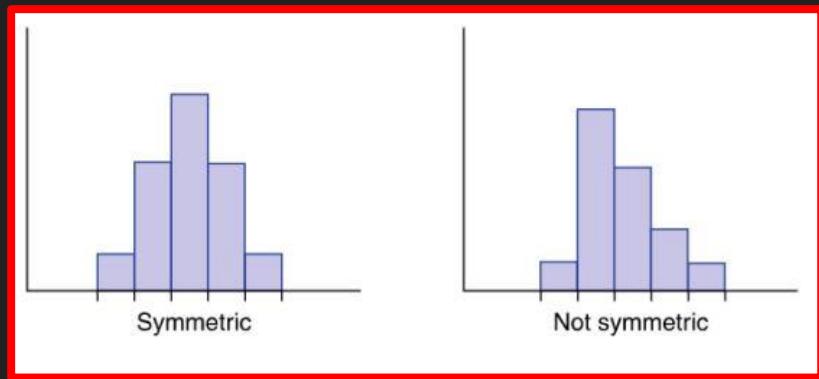
- When unique values are numerous
- Use Bins or Class Intervals

```
171, 174, 178, 181, 183, 183, 183, 184, 187, 188, 191,  
191, 191, 191, 192, 193, 193, 193, 194, 194, 195  
196, 196, 199, 200, 200, 200, 202, 204, 204, 205, 206,  
206, 211, 212, 213, 213, 216, 220, 221, 221, 227
```

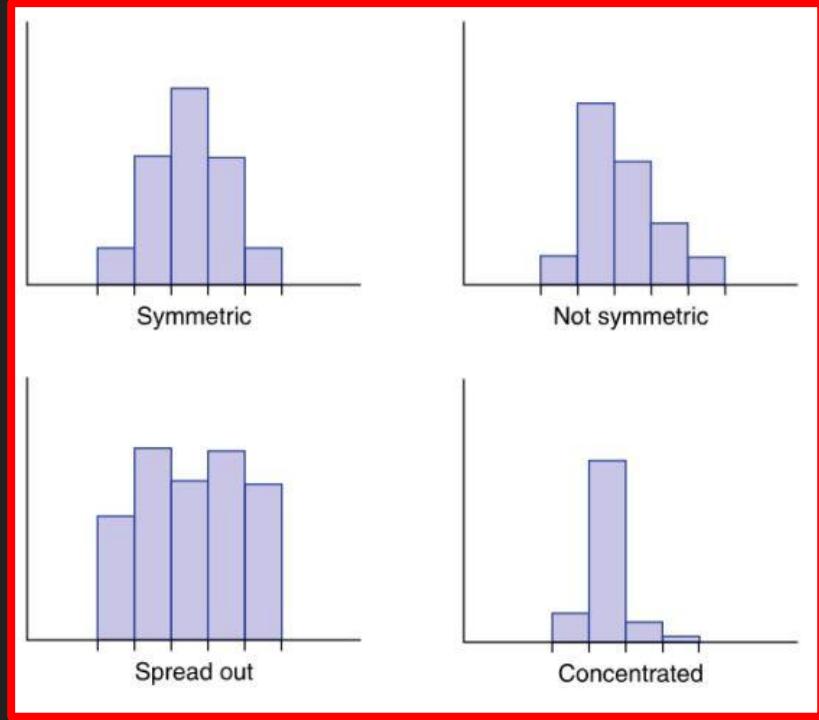
Sorted Blood cholesterol levels



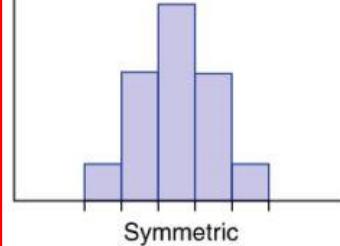
Histogram Types



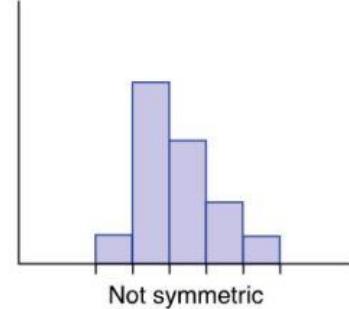
Histogram Types



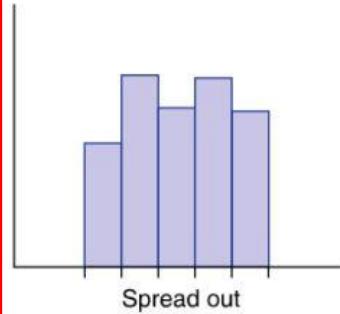
Histogram Types



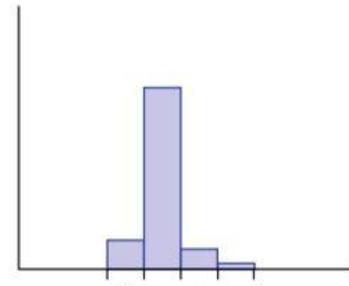
Symmetric



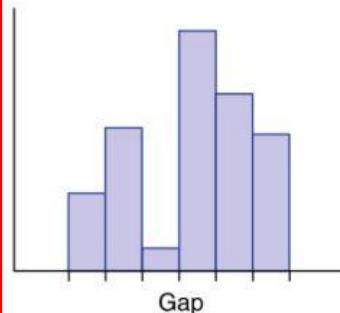
Not symmetric



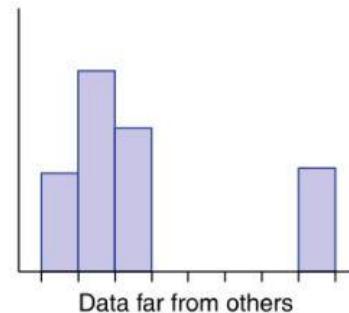
Spread out



Concentrated

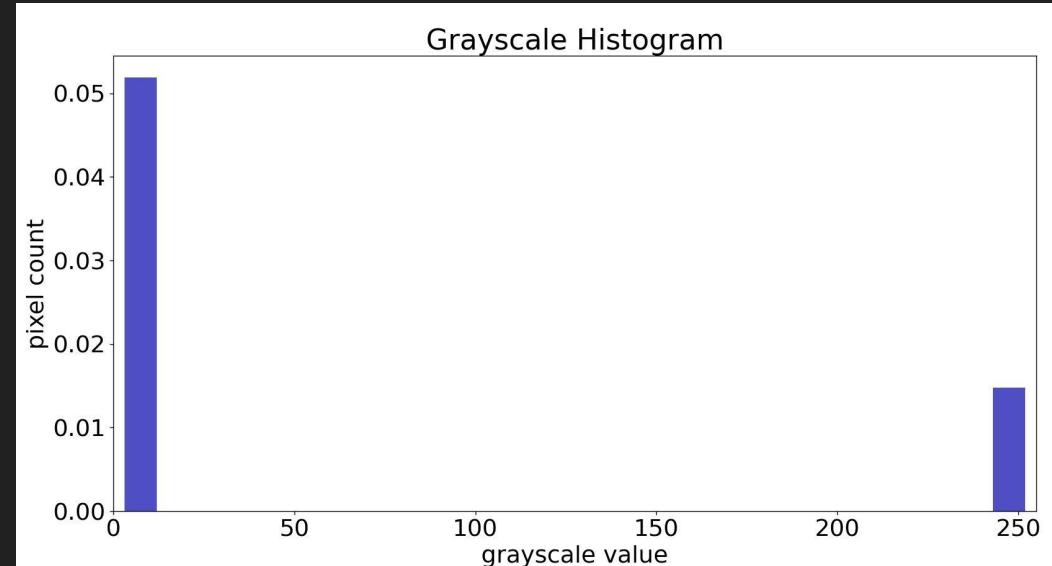
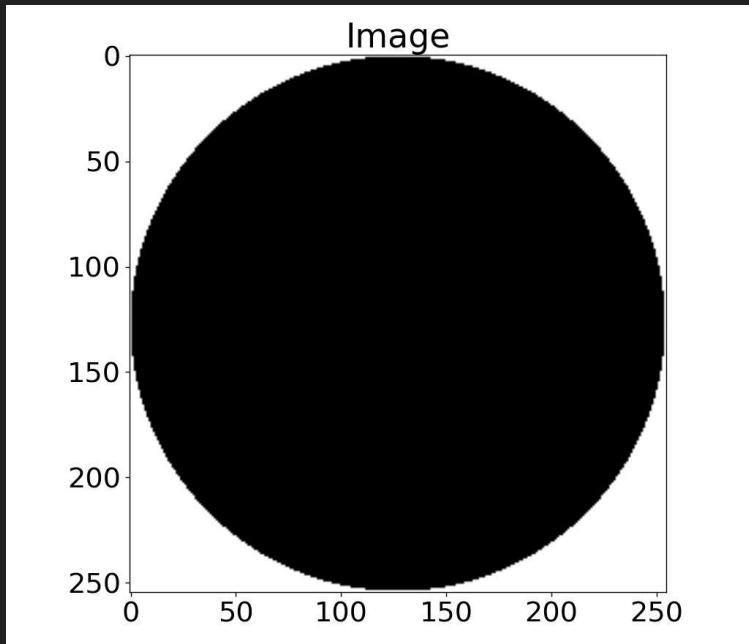


Gap

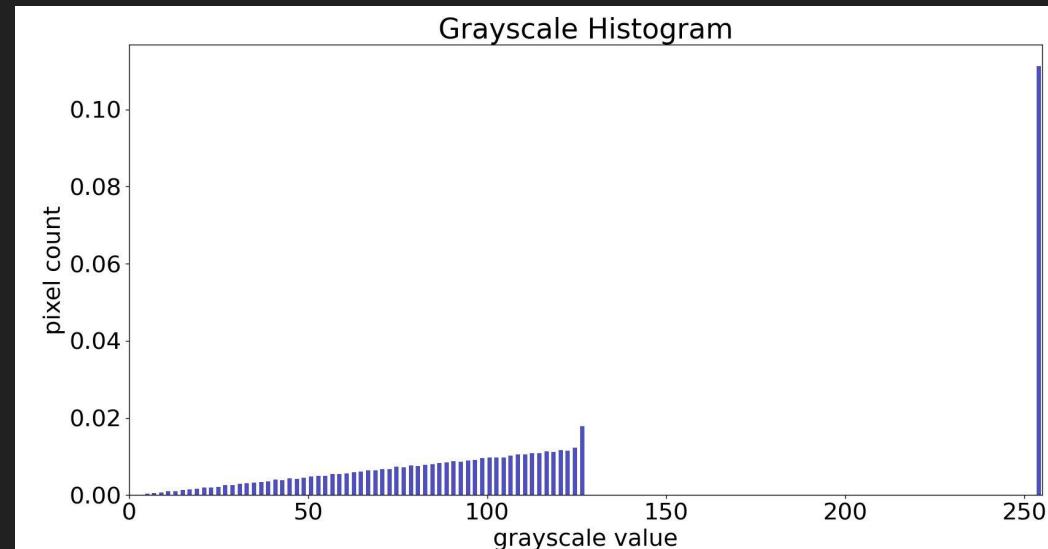
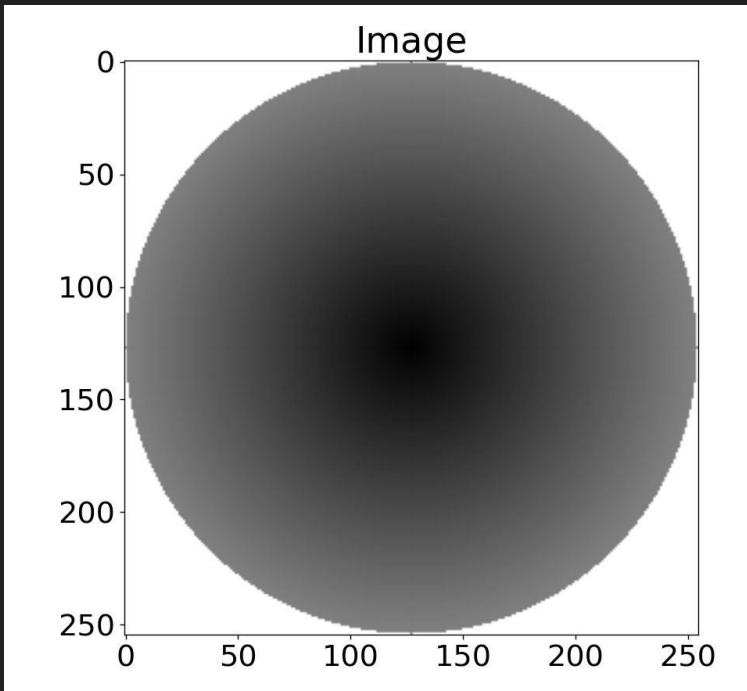


Data far from others

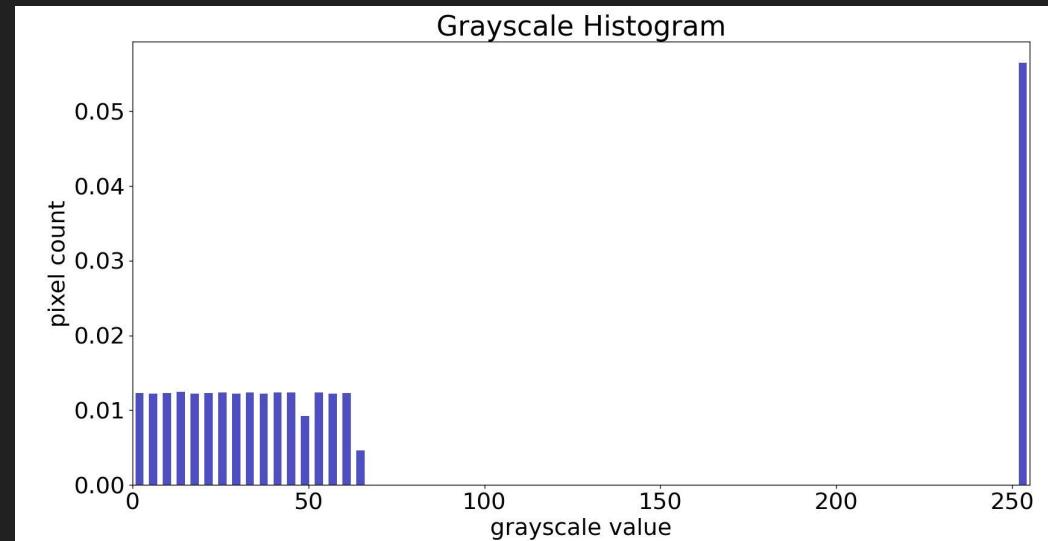
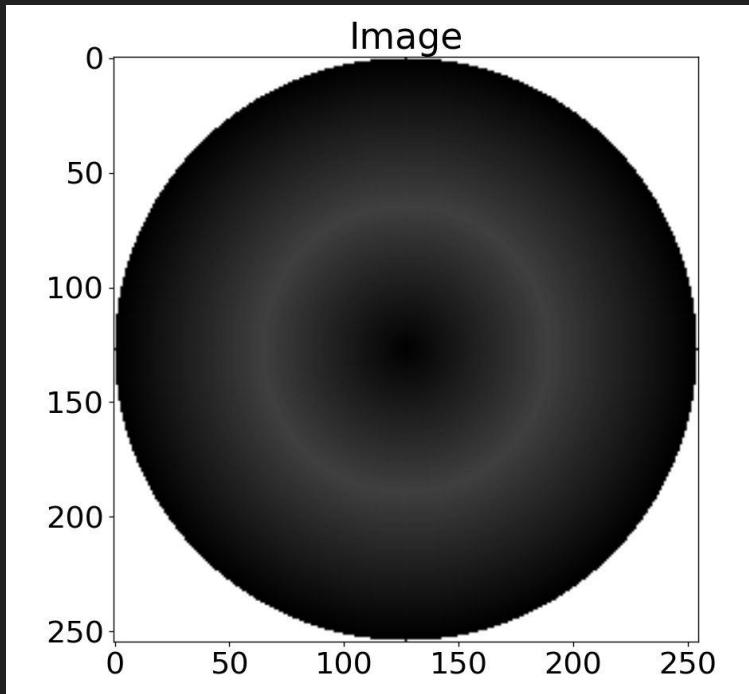
Some Images and Histograms



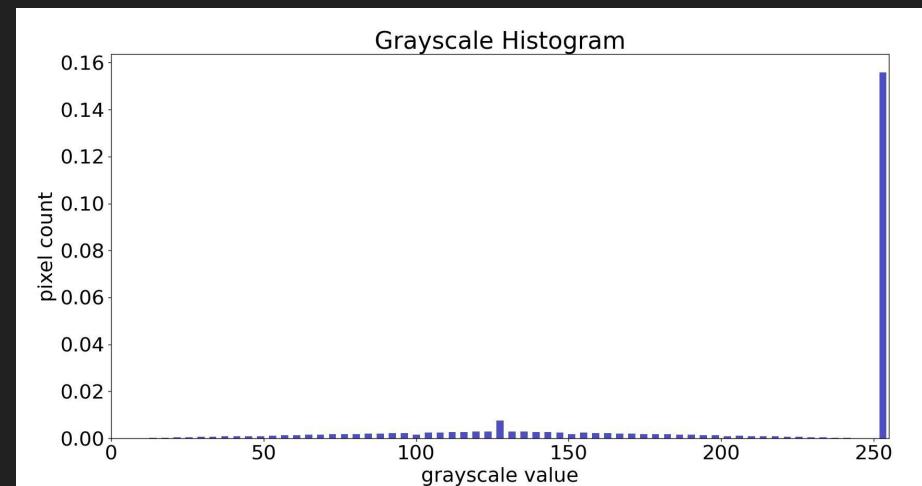
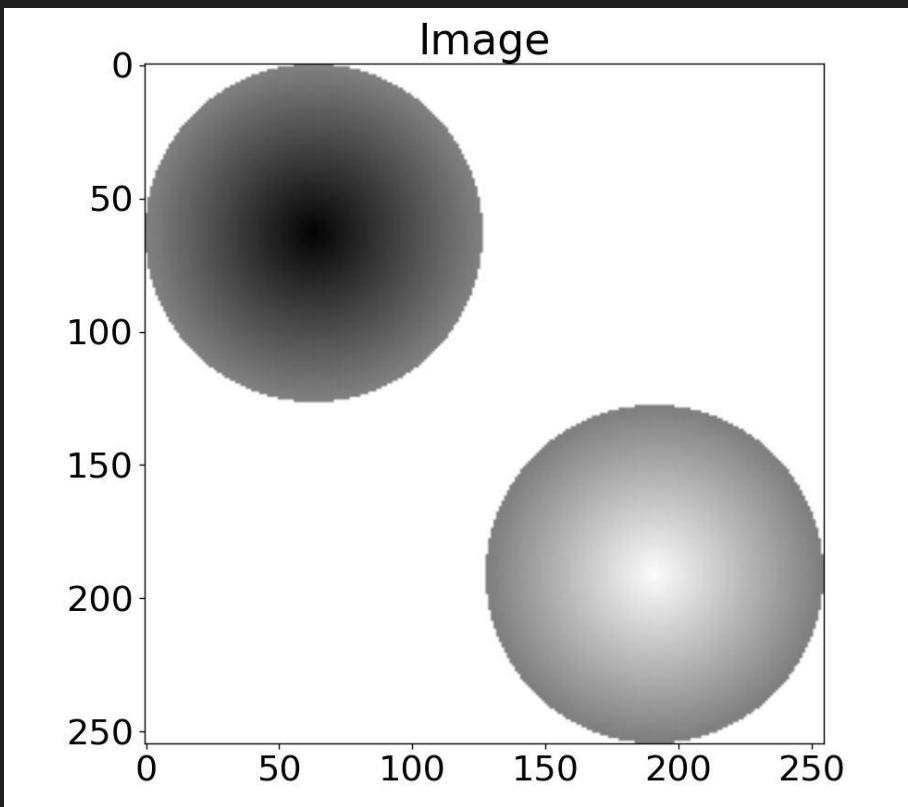
Some Images and Histograms



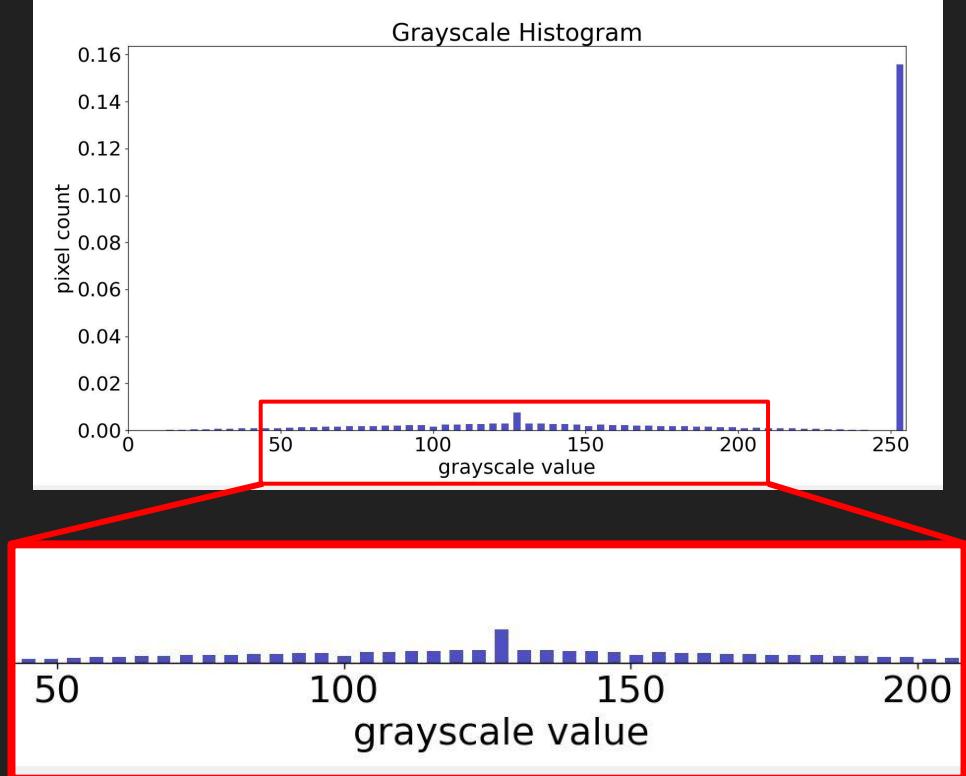
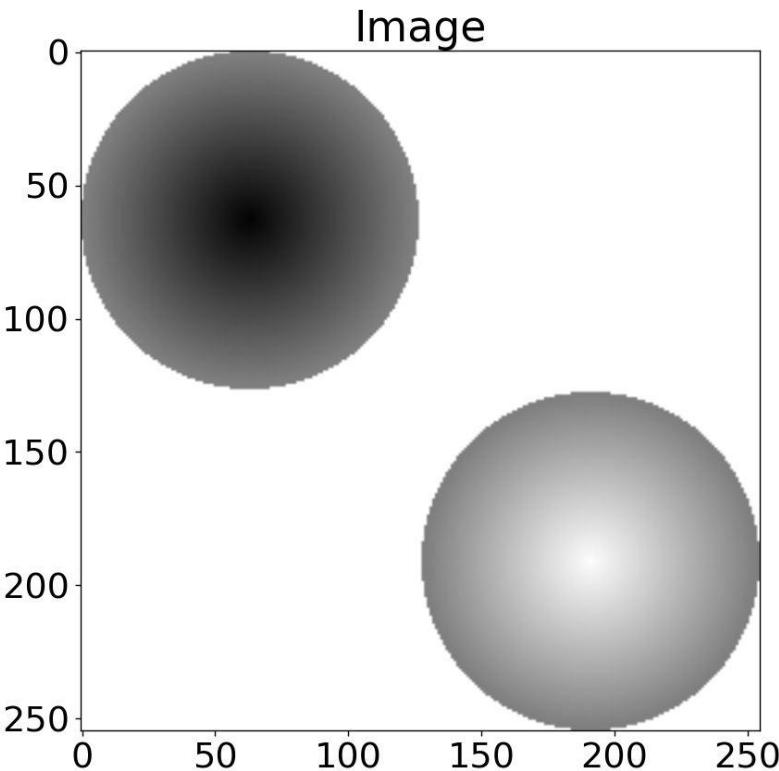
Some Images and Histograms



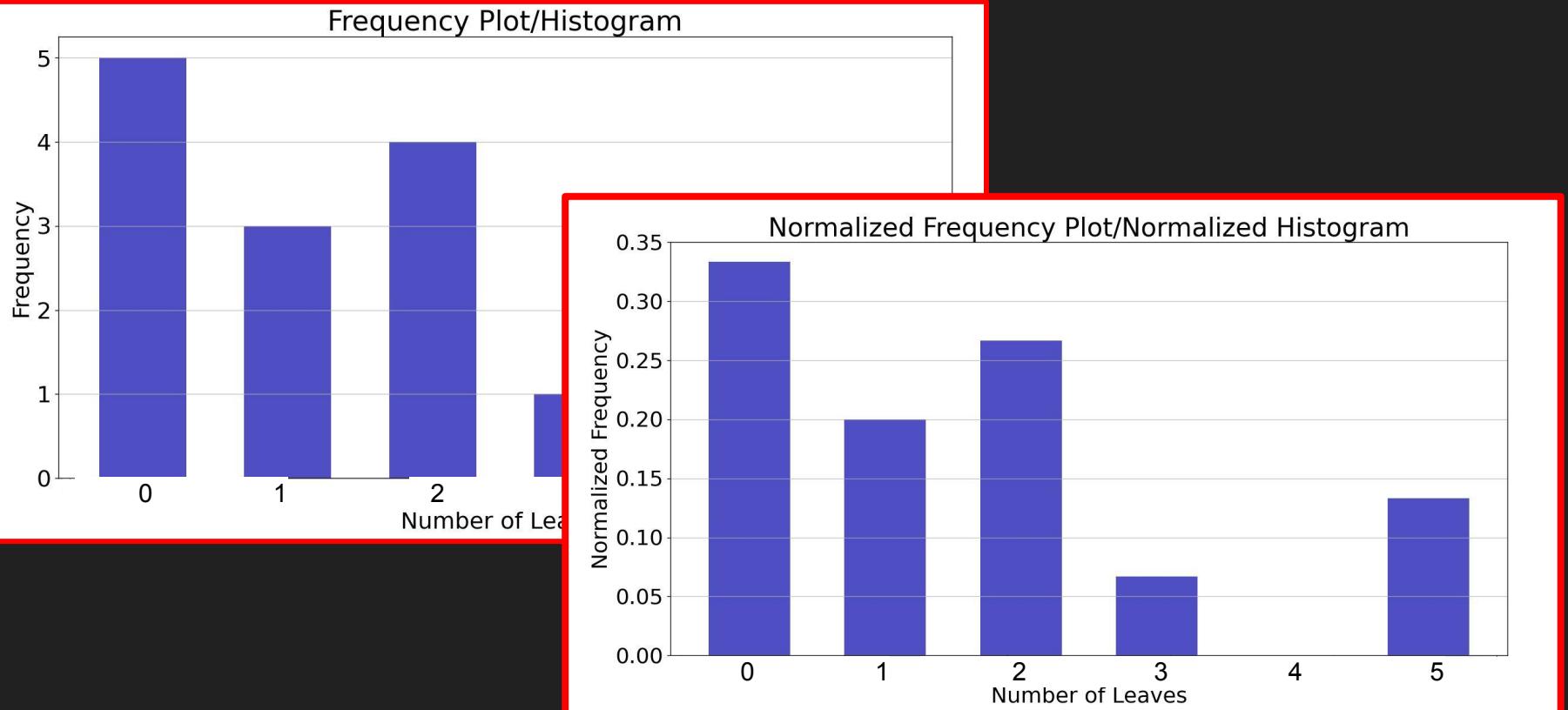
Some Images and Histograms



Some Images and Histograms



Describing Datasets: Why Normalization?



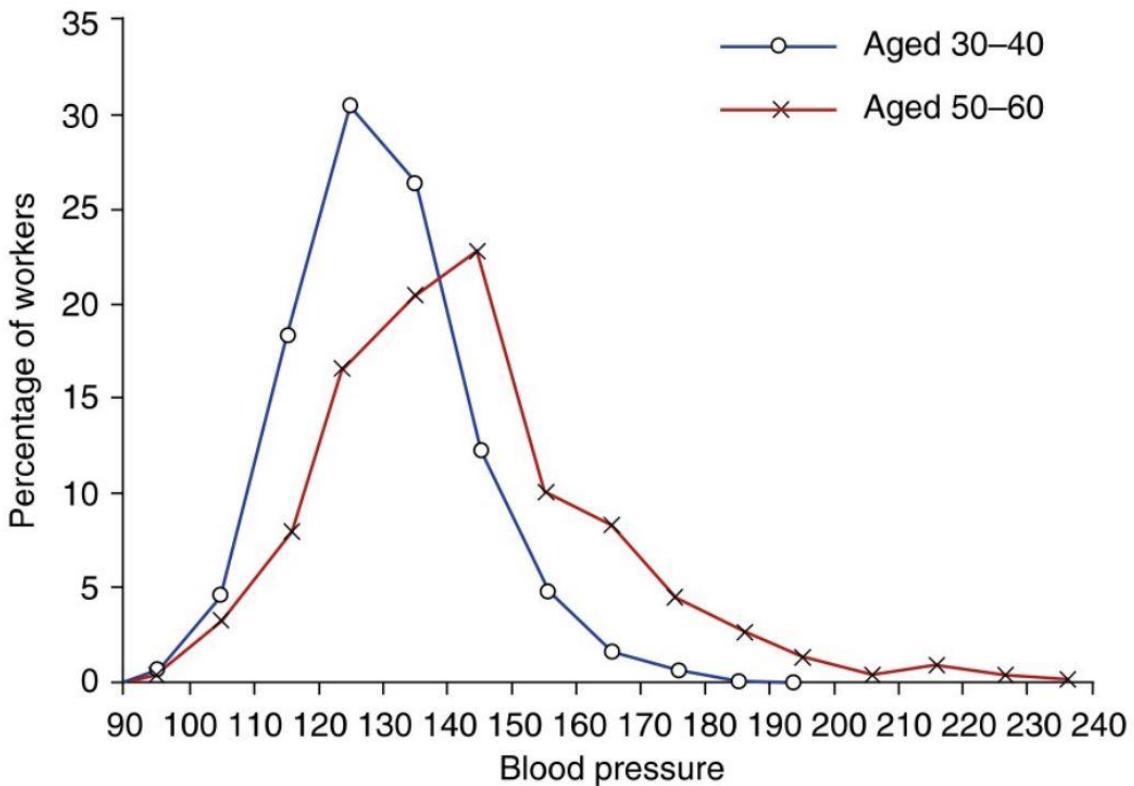
Describing Datasets: Why Normalization?

Blood pressure	Number of workers	
	Aged 30–40	Aged 50–60
Less than 90	3	1
90–100	17	2
100–110	118	23
110–120	460	57
120–130	768	122
130–140	675	149
140–150	312	167
150–160	120	73
160–170	45	62
170–180	18	35
180–190	3	20
190–200	1	9
200–210		3
210–220		5
220–230		2
230–240		1
Total	2540	731

Number of samples are unequal

Blood pressure	Percentage of workers	
	Aged 30–40	Aged 50–60
Less than 90	0.12	0.14
90–100	0.67	0.27
100–110	4.65	3.15
110–120	18.11	7.80
120–130	30.24	16.69
130–140	26.57	20.38
140–150	12.28	22.84
150–160	4.72	9.99
160–170	1.77	8.48
170–180	0.71	4.79
180–190	0.12	2.74
190–200	0.04	1.23
200–210		0.41
210–220		0.68
220–230		0.27
230–240		0.14
Total	100.00	100.00

Describing Datasets: Frequency Polygons

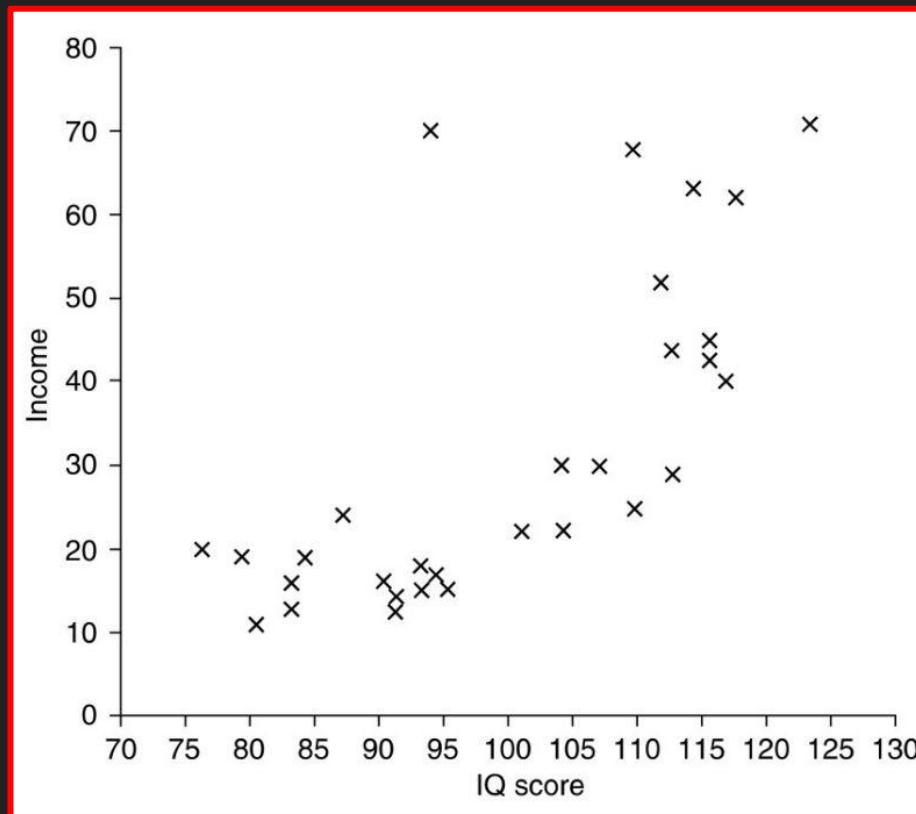


Relative
Frequency Polygons

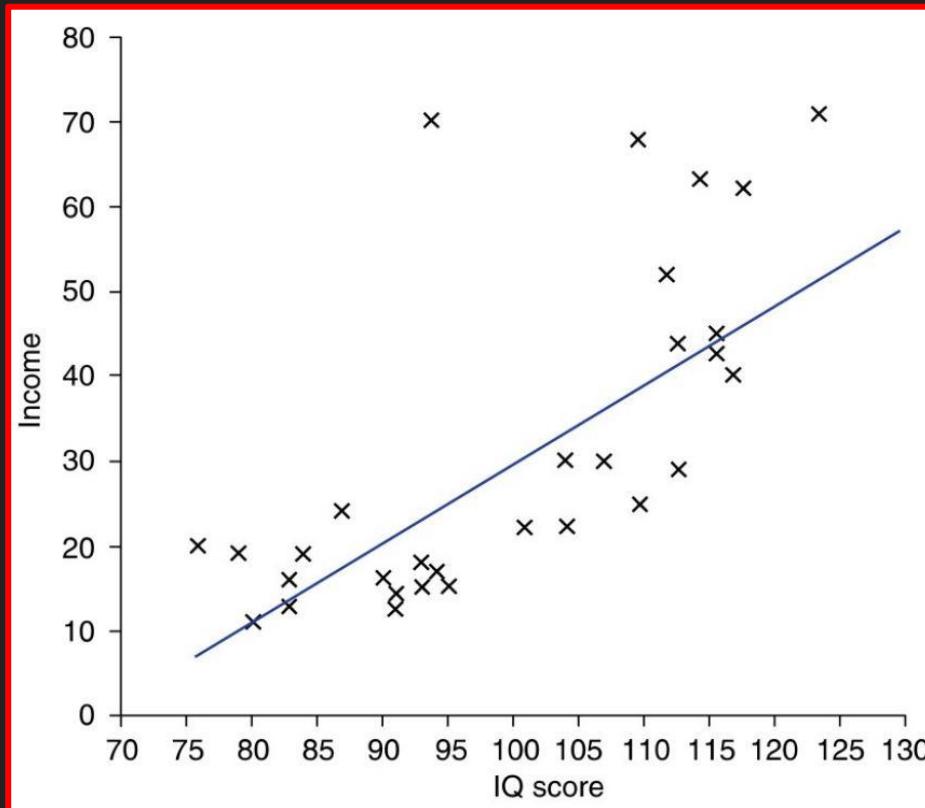
Describing Datasets: Scatter Plot

Worker i	IQ score x_i	Annual salary y_i (in units of \$1000)	Worker i	IQ score x_i	Annual salary y_i (in units of \$1000)
1	110	68	16	84	19
2	107	30	17	83	16
3	83	13	18	112	52
4	87	24	19	80	11
5	117	40	20	91	13
6	104	22	21	113	29
7	110	25	22	124	71
8	118	62	23	79	19
9	116	45	24	116	43
10	94	70	25	113	44
11	93	15	26	94	17
12	101	22	27	95	15
13	93	18	28	104	30
14	76	20	29	115	63
15	91	14	30	90	16

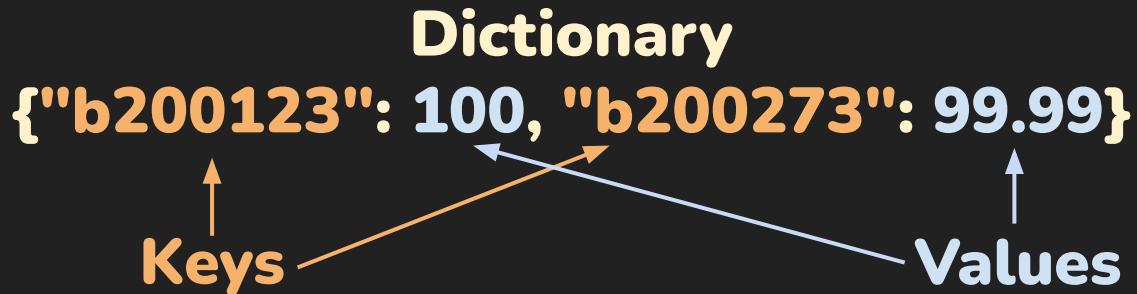
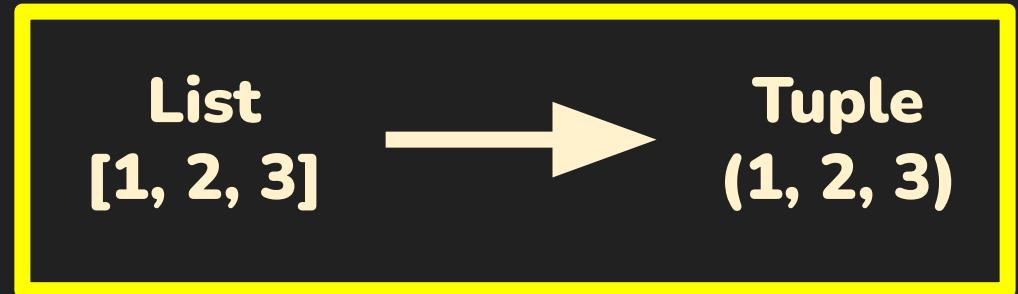
Describing Datasets: Scatter Plot



Describing Datasets: Scatter Plot



Tuples and Dictionaries



Tuples

- Tuple: An ordered collection of values
- Eg. (x,y,z)
 - coordinates in 3-D space
 - Width, Height, and Number of Channels, i.e. (W, H, C) of images in a dataset

```
>>> myTuple = (64, 64, 3)
```

Tuples

- Tuple: An ordered collection of values
- Eg. (x,y,z)
 - coordinates in 3-D space
 - Width, Height, and Number of Channels, i.e. (W, H, C) of images in a dataset

```
>>> myTuple = (64, 64, 3)
>>> print(myTuple[2])
3
```

Tuples

- Tuple: An ordered collection of values
- Eg. (x,y,z)
 - coordinates in 3-D space
 - Width, Height, and Number of Channels, i.e. (W, H, C) of images in a dataset

```
>>> myTuple = (64, 64, 3)
>>> print(myTuple[2])
3
>>> print(myTuple[1:])
(64, 3)
```

Tuples

- Tuple: An ordered collection of values
- Eg. (x,y,z)
 - coordinates in 3-D space
 - Width, Height, and Number of Channels, i.e. (W, H, C) of images in a dataset

```
>>> myTuple = (64, 64, 3)
>>> print(myTuple[2])
3
>>> print(myTuple[1:])
(64, 3)
>>> myTuple[1] = 128
Traceback (most recent call last):
  File "/usr/lib/python3.10/idlelib/run.py", line 578, in runcode
    exec(code, self.locals)
  File "<pyshell#3>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuples

```
>>> singleton = 'hello',  
>>> print(singleton)  
('hello',)
```

Tuples

```
>>> singleton = 'hello',  
>>> print(singleton)  
('hello',)  
>>> emptyTuple = ()  
>>> print(len(emptyTuple))  
0
```

Tuples

```
>>> singleton = 'hello',  
>>> print(singleton)  
('hello',)  
>>> emptyTuple = ()  
>>> print(len(emptyTuple))  
0  
>>> print(len(singleton))  
1
```

Tuples

```
>>> singleton = 'hello',
>>> print(singleton)
('hello',)
>>> emptyTuple = ()
>>> print(len(emptyTuple))
0
>>> print(len(singleton))
1
>>> print(type(emptyTuple))
<class 'tuple'>
```

Tuples

```
>>> singletons = 'hello', 'hi',
>>> print(type(singletons), len(singletons))
<class 'tuple'> 2
```

Tuples

- `len()`
- `max()`
 - `max('Hi', 'hi')`
 - `max(2, 'hi')`
- `min()`
- `sum()`

Tuples

- `any((0, False))`
- `any('0', False))`
- `all()`
- `srt = sorted(['a', 'A', 'b', 'B'])`
- `print(srt)`
- `['A', 'B', 'a', 'b']`
- `tuple()`
- `print(tuple('string'))`
- `('s', 't', 'r', 'i', 'n', 'g')`
- `print(tuple(list(range(5))))`
- `(0, 1, 2, 3, 4)`

Tuples and Lists

- a = (1, 3 , 5, 3, 1)
- a.index(3)
- a.count(3)
- You can loop into tuples like lists.
- You can use “in” and “not in” operators like lists.
- You can concatenate tuples with “+” operators like lists.
- (1,2,3)<(4,5,6)
- True
- [1,2,3]<[4,5,6]
- True

Tuples and Lists

- `a = (0,1)`
- `(0,1)` is a
- `False`
- Two tuples or lists have different identity
- `(0,1) == a`
- `True`
- `b = a`
- `b` is a
- `True`
- `c = (0,1)`
- `c` is a
- `False`

Tuples and Lists

- $(0,1)^{(3,4)}$
- $[0,1]^*[3,4]$
- $[0,1,2]*[3,4,5]$
- $[0,1,2]**[3,4,5]$
- $(0,1,2)**(3,4,5)$
- How to perform element-wise operations?

Tuples

- Tuples are faster than lists
 - Immutable
- Tuples are used with dictionaries

```
nested = [1 , (2, 3, 4), [5, 6, 7, 8, 9]]  
len(nested)  
3  
nested[1][2]  
4  
nested[1][1]  
3  
newNested = (1 , [2, 3, 4], (5, 6, 7, 8, 9))
```

Tuples and Dictionaries

List
[1, 2, 3]



Tuple
(1, 2, 3)

Dictionary

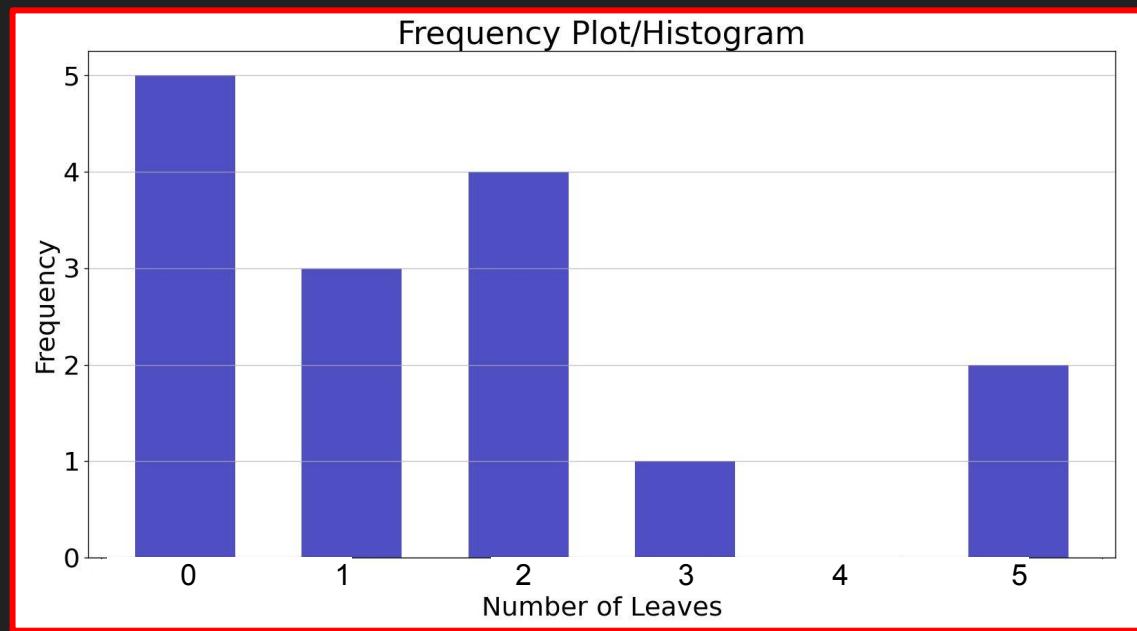
{"b200123": 100, "b200273": 99.99}

Keys

Values

Dictionaries: Motivation

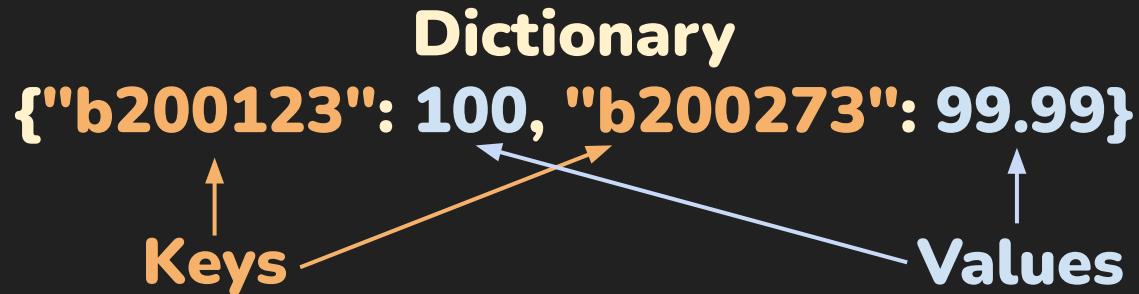
How to spot errors in words or find erroneous words like
Tableeeee?



Dictionaries

Dictionaries

- are like lists
- except that the index is any immutable type
- key:value pairs
- aka associative arrays or hashables
- store a value and retrieve it
- are mutable



Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}  
  
#Retrieve a value with dictName[keyName]:  
print(extension['Shyam'])
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}  
  
#Retrieve a value with dictName[keyName]:  
print(extension['Shyam'])  
#3582
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}  
  
#Retrieve a value with dictName[keyName]:  
print(extension['Shyam'])  
#3582  
  
del extension['Shyam']  
print(extension)  
#{'Ram': 4312, 'Mohan': 5672}
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}  
  
#Retrieve a value with dictName[keyName]:  
print(extension['Shyam'])  
#3582  
  
del extension['Shyam']  
print(extension)  
#{'Ram': 4312, 'Mohan': 5672}  
  
extensionList = list(extension)  
print(extensionList)  
#[ 'Ram', 'Mohan' ]
```

Dictionaries

```
#Defining a dictionary:  
extension = {'Ram': 4312, 'Shyam': 3582}  
#Add a new item or store a new value:  
extension['Mohan'] = 5672  
print(type(extension))  
#<class 'dict'>  
print(extension)  
#{'Ram': 4312, 'Shyam': 3582, 'Mohan': 5672}  
  
#Retrieve a value with dictName[keyName]:  
print(extension['Shyam'])  
#3582  
  
del extension['Shyam']  
print(extension)  
#{'Ram': 4312, 'Mohan': 5672}  
  
extensionList = list(extension)  
print(extensionList)  
#[('Ram', 'Mohan')]  
  
print('Ram' in extension)  
#True  
print('Shyam' in extension)  
#False
```

Dictionaries: Key Error

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

extension['Shyam']
```

```
{'Ram': 4312, 'Mohan': 5672}
Traceback (most recent call last):
  File "D:/0000Mandi/Lec14 Scripts/dicts.py", line 30, in <module>
    extension['Shyam']
KeyError: 'Shyam'
```

Dictionaries: Methods

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

print(extension.keys())
# dict_keys(['Ram', 'Mohan'])
```

Dictionaries: Methods

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

print(extension.keys())
# dict_keys(['Ram', 'Mohan'])

print(extension.values())
# dict_values([4312, 5672])

print(extension.items())
# dict_items([('Ram', 4312), ('Mohan', 5672)])
```

Dictionaries: Methods

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

print(extension.keys())
# dict_keys(['Ram', 'Mohan'])

print(extension.values())
# dict_values([4312, 5672])

print(extension.items())
# dict_items([('Ram', 4312), ('Mohan', 5672)])

#Avoiding key error:
print(extension.get('Ram', 'not found'))
```

Dictionaries: Methods

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

print(extension.keys())
# dict_keys(['Ram', 'Mohan'])

print(extension.values())
# dict_values([4312, 5672])

print(extension.items())
# dict_items([('Ram', 4312), ('Mohan', 5672)])

#Avoiding key error:
print(extension.get('Ram', 'not found'))
#4312

print(extension.get('Shyam', 'not found'))
#not found
```

Dictionaries:

Remember

- Cannot have multiple items with same key
- Keys have to be immutable
- Values need not be unique, they can be mutable or immutable

Dictionaries:

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

extension['Ram'] = 3211
print(extension)
#{'Ram': 3211, 'Mohan': 5672}
```

Dictionaries:

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

extension['Ram'] = 3211
print(extension)
#{'Ram': 3211, 'Mohan': 5672}

myList = [1 , 2, 3]
extension[myList] = 4312
```

Dictionaries:

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

extension['Ram'] = 3211
print(extension)
#{'Ram': 3211, 'Mohan': 5672}

myList = [1 , 2, 3]
extension[myList] = 4312
'''
Traceback (most recent call last):
  File "D:/0000Mandi/Lec14 Scripts/dicts.py", line 59, in <module>
    extension[myList] = 4312
TypeError: unhashable type: 'list'
'''
```

Dictionaries:

```
print(extension)
#{'Ram': 4312, 'Mohan': 5672}

extension['Ram'] = 3211
print(extension)
#{'Ram': 3211, 'Mohan': 5672}

myList = [1 , 2, 3]
extension[myList] = 4312
'''
Traceback (most recent call last):
  File "D:/0000Mandi/Lec14 Scripts/dicts.py", line 59, in <module>
    extension[myList] = 4312
TypeError: unhashable type: 'list'
'''

extension['Arjun'] = myList
print(extension)
{'Ram': 3211, 'Mohan': 5672, 'Arjun': [1, 2, 3]}
```

List and Dictionary Comprehension:

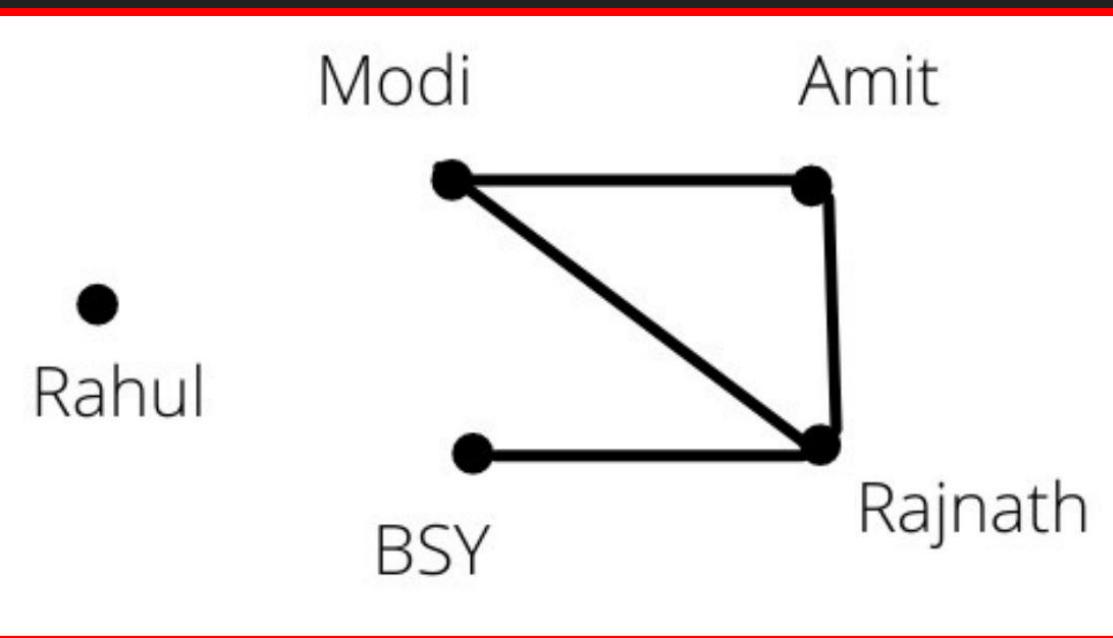
```
# List Comprehension
cubesList = [x**3 for x in range(5)]
print(cubesList)
# [0, 1, 8, 27, 64]
```

List and Dictionary Comprehension:

```
# List Comprehension
cubesList = [x**3 for x in range(5)]
print(cubesList)
# [0, 1, 8, 27, 64]

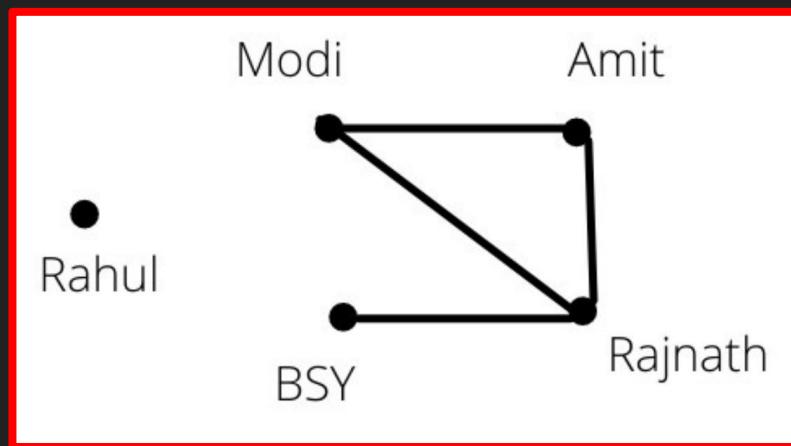
# Dict Comprehension
# cubesDict = [x: x**3 for x in range(5)] #be careful
cubesDict = {x: x**3 for x in range(5)}
print(cubesDict)
# {0: 0, 1: 1, 2: 8, 3: 27, 4: 64}
```

Dictionary Application



Friendship Graph

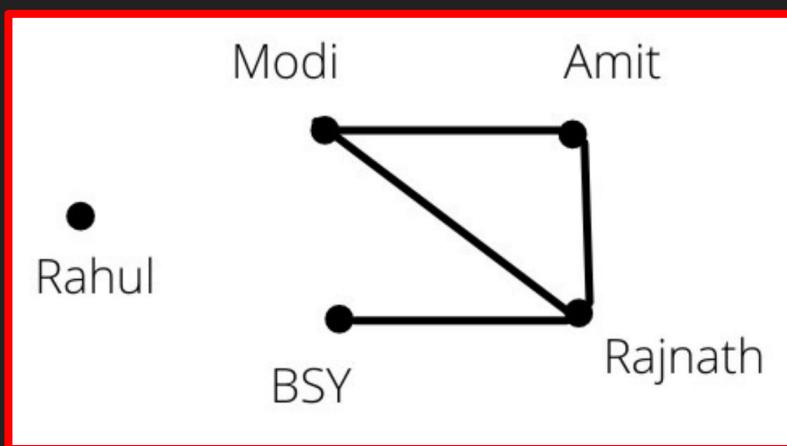
Dictionary Application



Friendship Graph

- Adjacency matrix representing friendship graph

Dictionary Application

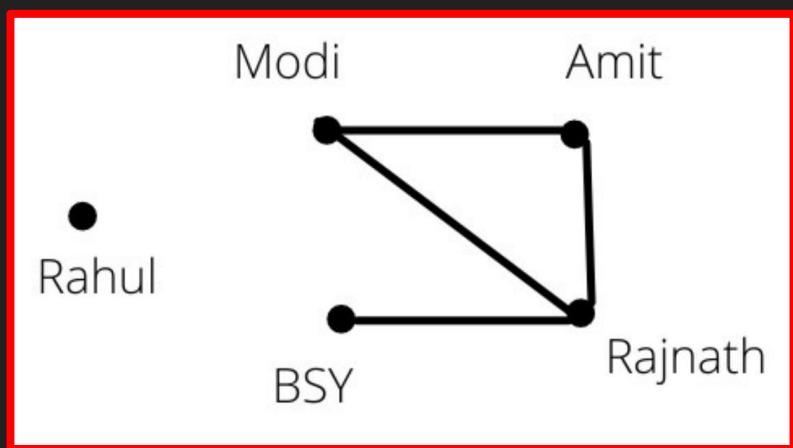


Friendship Graph

- Adjacency matrix representing friendship graph

	M	A	Rj	BS	Rh
M	1	1	1	0	0
A	1	1	1	0	0
Rj	1	1	1	1	0
BS	0	0	1	1	0
Rh	0	0	0	0	1

Dictionary Application



Friendship Graph

- Adjacency matrix representing friendship graph
- Sparse Matrix: lot of elements are zero

	M	A	Rj	BS	Rh
M	1	1	1	0	0
A	1	1	1	0	0
Rj	1	1	1	1	0
BS	0	0	1	1	0
Rh	0	0	0	0	1

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

Dictionary Application: Sparse Matrix

[0	0	0	1	0]
0	0	0	0	0		
0	2	0	0	0		
0	0	0	0	0		
0	0	0	3	0		

```
matrix = [ [0,0,0,1,0],  
          [0,0,0,0,0],  
          [0,2,0,0,0],  
          [0,0,0,0,0],  
          [0,0,0,3,0] ]
```

List of Lists

Disadvantage?

Dictionary Application: Sparse Matrix

0	0	0	1	0
0	0	0	0	0
0	2	0	0	0
0	0	0	0	0
0	0	0	3	0

```
# listOfLists = [[0]*5]*5
```

Dictionary Application: Sparse Matrix

0	0	0	1	0
0	0	0	0	0
0	2	0	0	0
0	0	0	0	0
0	0	0	3	0

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1 , (2, 1): 2, (4, 3):3}
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1 , (2, 1): 2, (4, 3):3}

print(matrix[0,3])
# 1
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1, (2, 1): 2, (4, 3):3}

print(matrix[0,3])
# 1
print(matrix.get((4, 3), 0))
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1, (2, 1): 2, (4, 3):3}

print(matrix[0,3])
# 1
print(matrix.get((4, 3), 0))
# 3
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1, (2, 1): 2, (4, 3):3}

print(matrix[0,3])
# 1
print(matrix.get((4, 3), 0))
# 3
print(matrix.get((2, 3), 0))
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1, (2, 1): 2, (4, 3):3}

print(matrix[0,3])
# 1
print(matrix.get((4, 3), 0))
# 3
print(matrix.get((2, 3), 0))
# 0
```

Dictionary Application: Sparse Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

```
# listOfLists = [[0]*5]*5
# listOfLists[0][3] = 1
# listOfLists[2][1] = 2
# listOfLists[4][3] = 3
matrix = {(0, 3):1, (2, 1): 2, (4, 3):3}

print(matrix[0, 3])
# 1
print(matrix.get((4, 3), 0))
# 3
print(matrix.get((2, 3), 0))
# 0
```

HW: implement sparse matrix operations using a dictionary: add, scale, trace. Use provided skeleton code

Visualization in Python

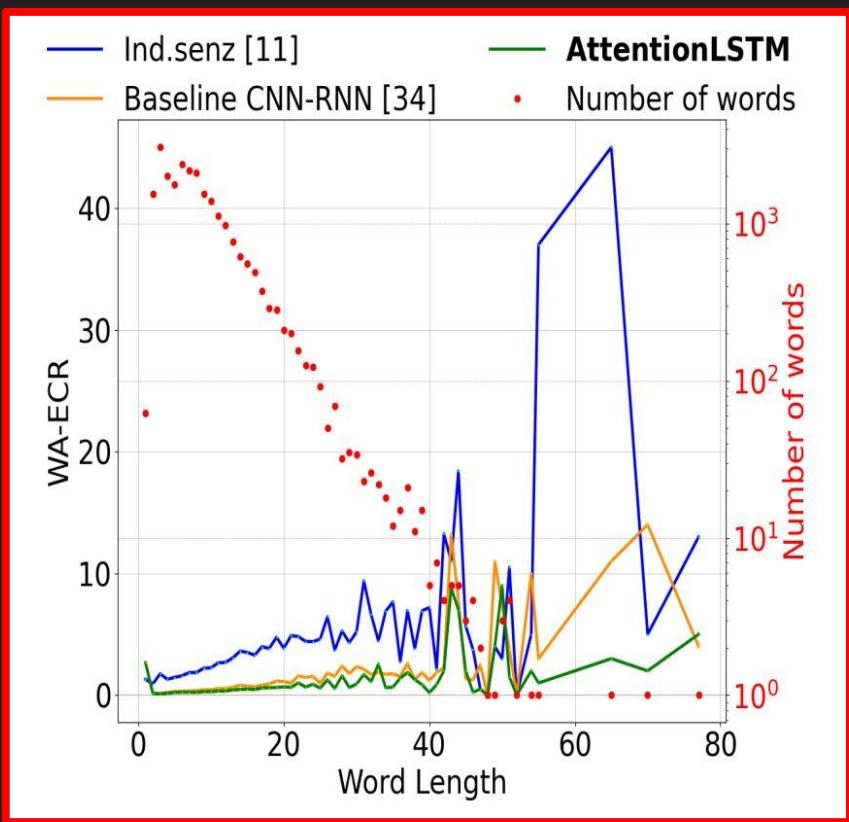


Figure Anatomy

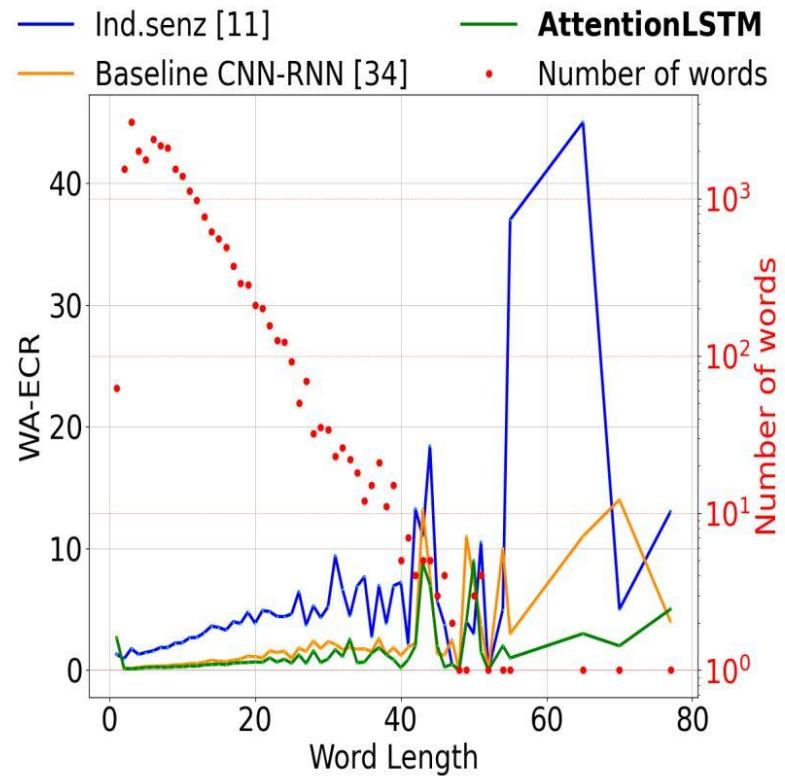
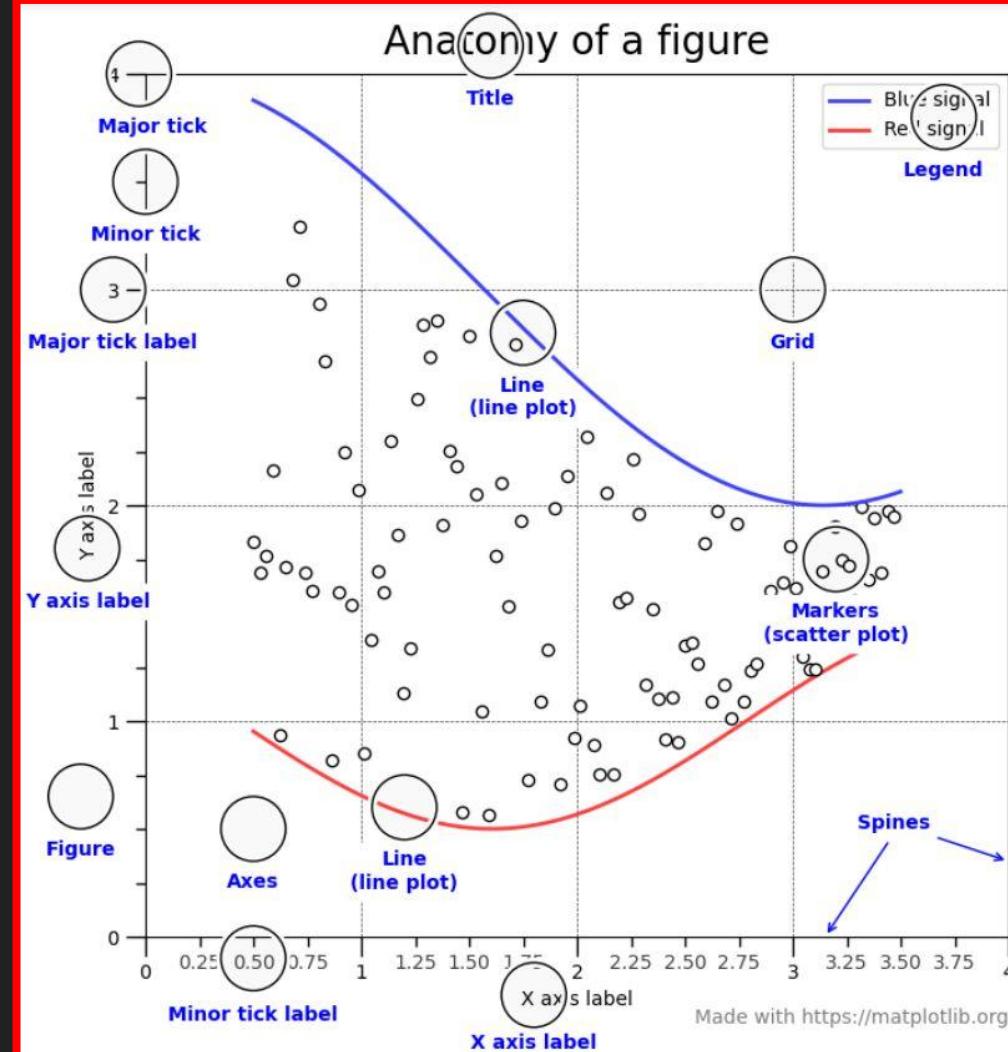
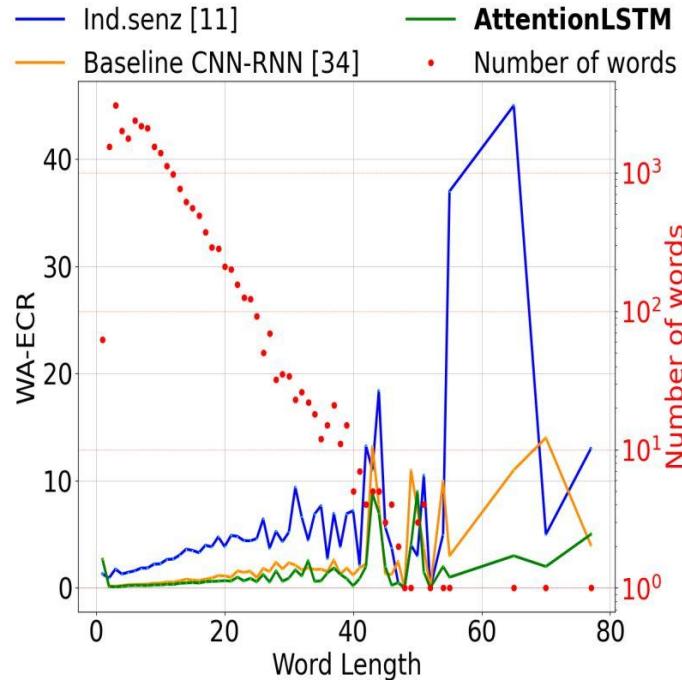
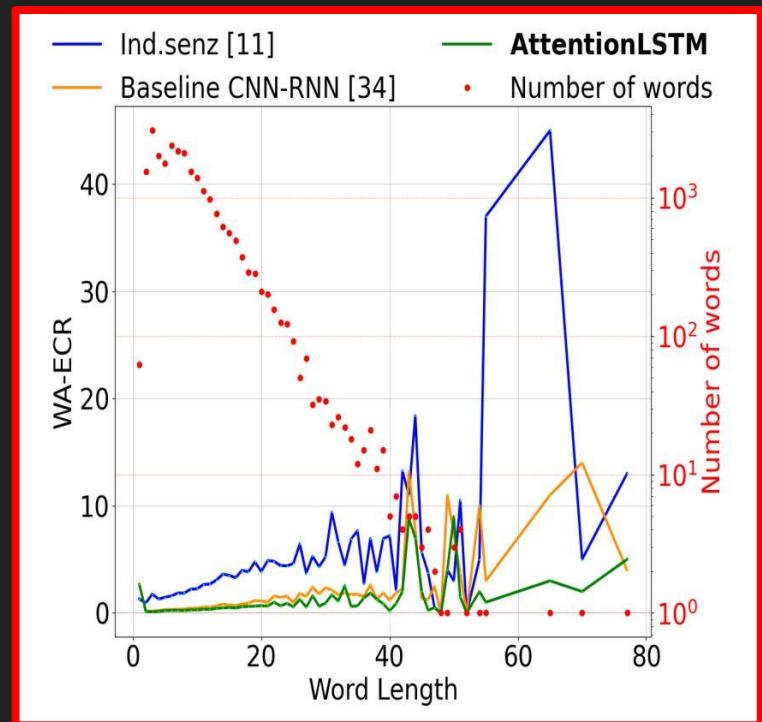


Figure Anatomy

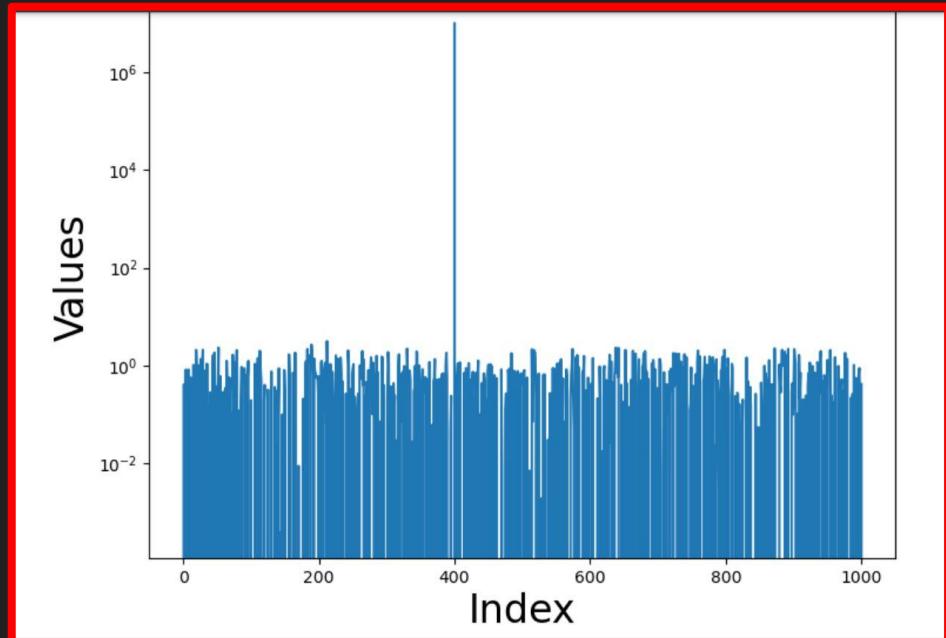
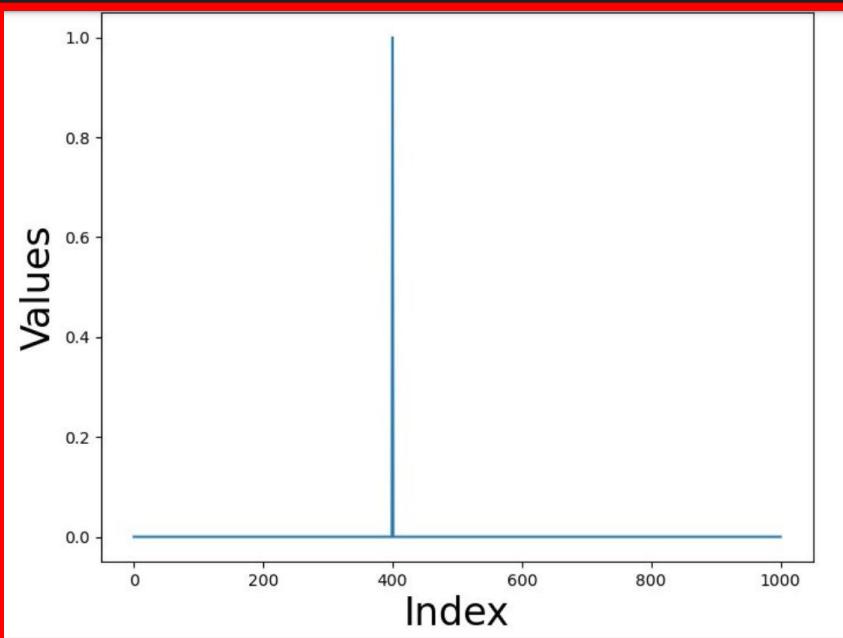


Plotting dos and don'ts

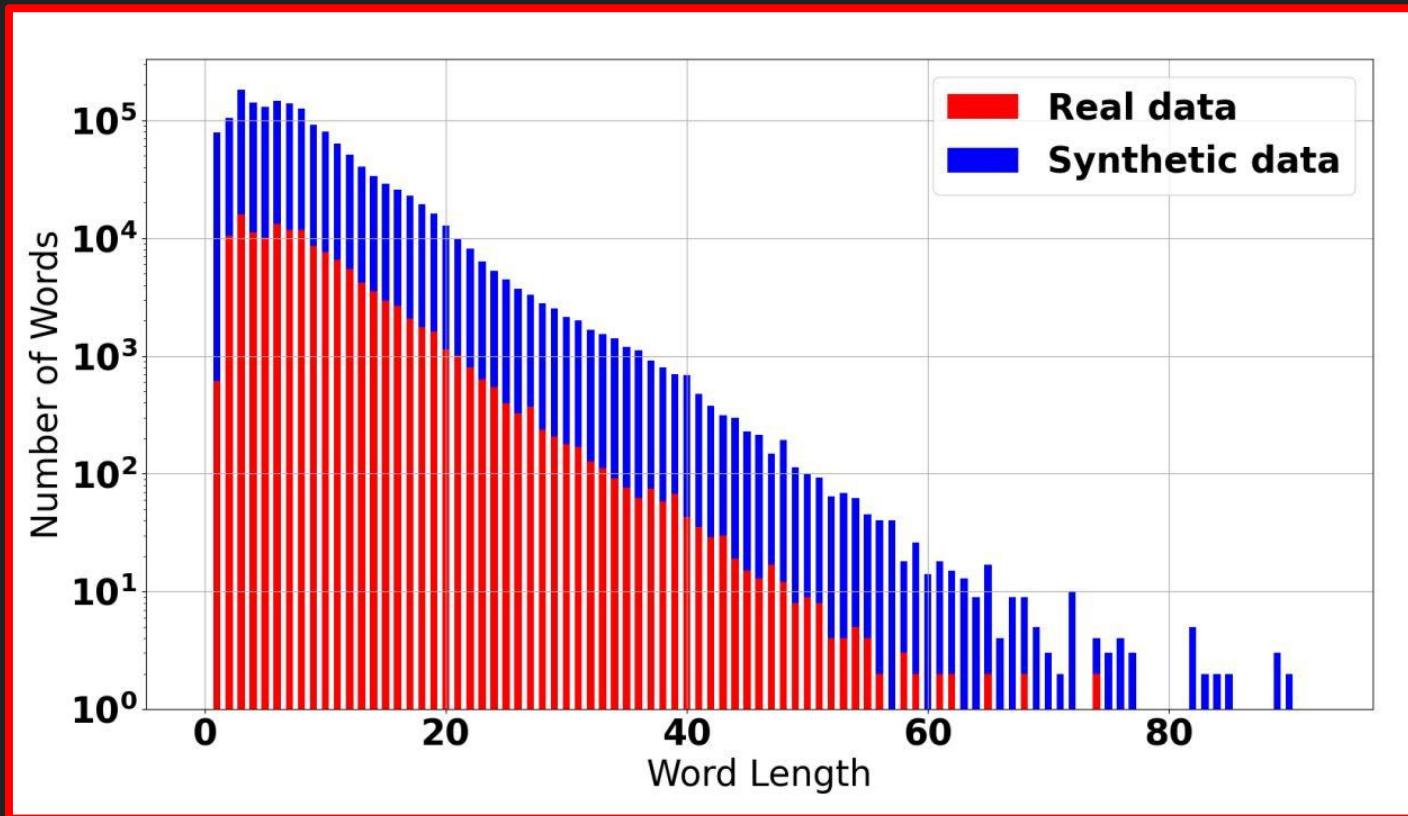
- Axis ranges: must be common if you want to compare figures
- Axis scale: linear or log, based on range of values
- Axis labels: always label the axis.
- Font: Use font size which is clearly visible.
- Give name and units. e.g. number, length, response time (sec)
- Axis tick labels: format it right eg. 0.00000034m vs 34nm, 100000 vs 10^5



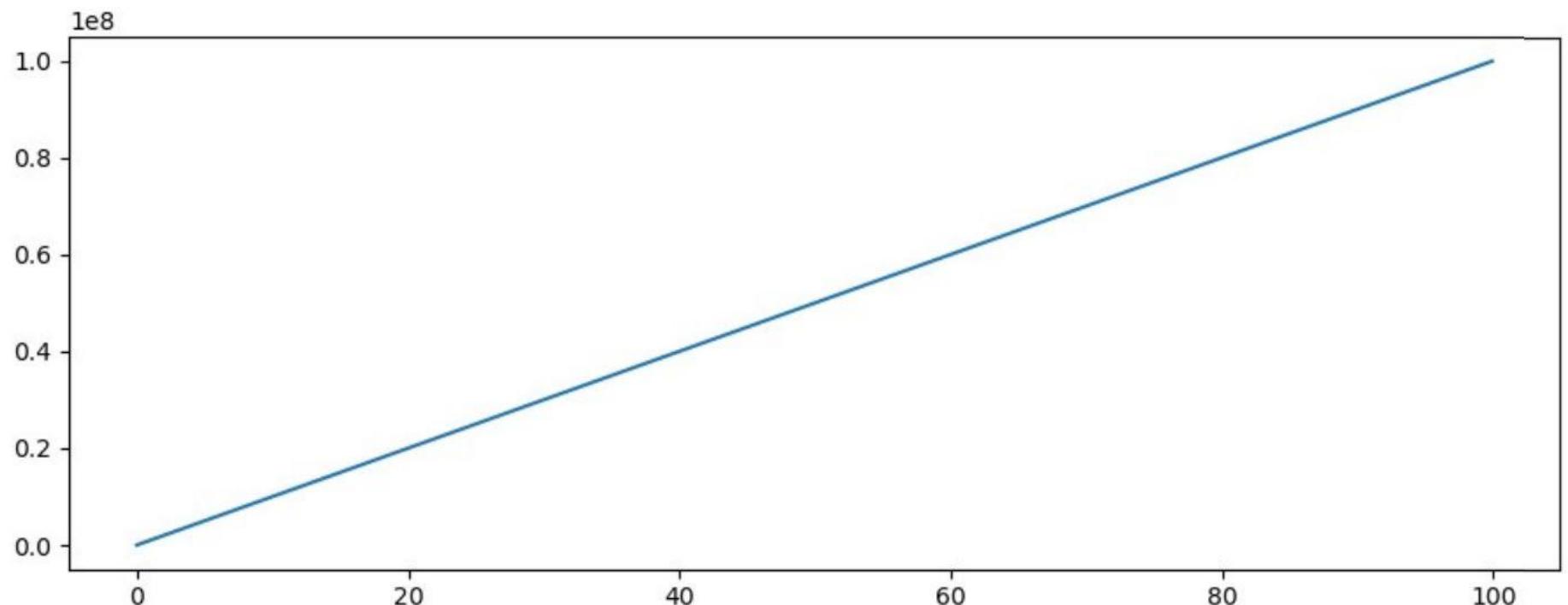
Scale



Scale



Scale



Misleading axis scales

Plotting Sine Wave

```
import math  
import matplotlib.pyplot as plt
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
```

We wanted to use values between -10π to 10π , so multiplying values in range [-500, 500] with $\pi/50$

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))
plt.plot(xList, yList) # makes the line plot
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))
plt.plot(xList, yList) # makes the line plot
plt.grid(True) #turning on grid
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))
plt.plot(xList, yList) # makes the line plot
plt.grid(True) #turning on grid
plt.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
plt.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
plt.title('Sine Wave', fontsize = 36) #adding the title
```

Plotting Sine Wave

```
import math
import matplotlib.pyplot as plt

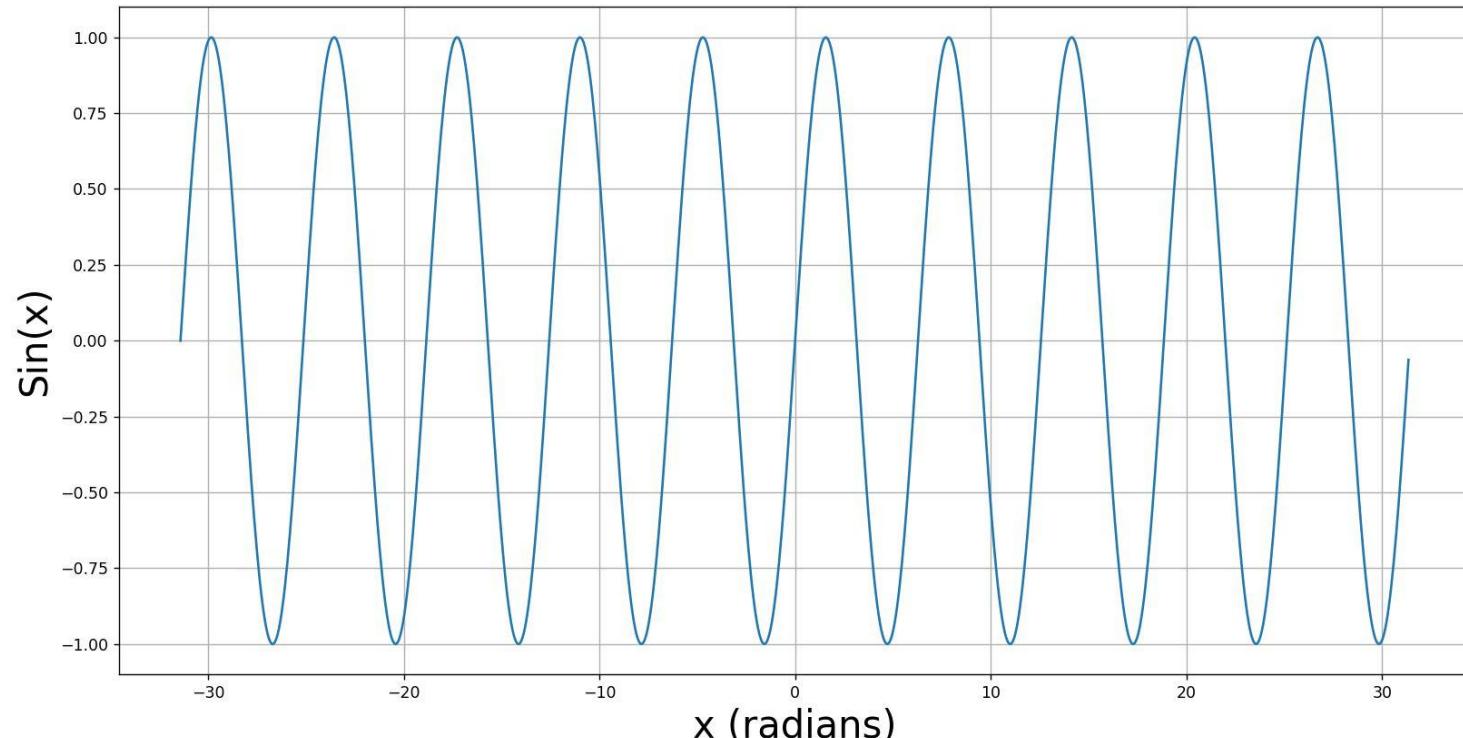
#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))
plt.plot(xList, yList) # makes the line plot
plt.grid(True) #turning on grid
plt.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
plt.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
plt.title('Sine Wave', fontsize = 36) #adding the title
plt.show() #you can only see plot if you use this statement
```

Plotting Sine Wave

Sine Wave



Plotting Sine Wave: Import Style 1

```
import math
import matplotlib.pyplot as plt

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))
plt.plot(xList, yList) # makes the line plot
plt.grid(True) #turning on grid
plt.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
plt.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
plt.title('Sine Wave', fontsize = 36) #adding the title
plt.show() #you can only see plot if you use this statement
```

Plotting Sine Wave: Import Style 2

```
import math
import matplotlib.pyplot

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))

matplotlib.pyplot.plot(xList, yList) # makes the line plot
matplotlib.pyplot.grid(True) #turning on grid
matplotlib.pyplot.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
matplotlib.pyplot.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
matplotlib.pyplot.title('Sine Wave', fontsize = 36) #adding the title
matplotlib.pyplot.show() #you can only see plot if you use this statement
```

Plotting Sine Wave: Import Style 2

```
import math
import matplotlib.pyplot

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))

matplotlib.pyplot.plot(xList, yList) # makes the line plot
matplotlib.pyplot.grid(True) #turning on grid
matplotlib.pyplot.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
matplotlib.pyplot.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
matplotlib.pyplot.title('Sine Wave', fontsize = 36) #adding the title
matplotlib.pyplot.show() #you can only see plot if you use this statement
```

Plotting Sine Wave: Import Style 3

```
import math
from matplotlib import pyplot

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))

pyplot.plot(xList, yList) # makes the line plot
pyplot.grid(True) #turning on grid
pyplot.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
pyplot.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
pyplot.title('Sine Wave', fontsize = 36) #adding the title
pyplot.show() #you can only see plot if you use this statement
```

Plotting Sine Wave: Import Style 3

```
import math
from matplotlib import pyplot

#picking 1000 points between -10 and 10, and multiplying with pi
xList = list(range(-500,500))#still need to multiply with (1/50) and pi

yList = []

#multiplying x with pi and 1/50, saving sine of the converted values
for i in range(len(xList)):
    xList[i] *= (math.pi * (1/50))
    yList.append(math.sin(xList[i]))
print(min(xList), max(xList))

pyplot.plot(xList, yList) # makes the line plot
pyplot.grid(True) #turning on grid
pyplot.xlabel('x (radians)', fontsize = 24) #adding label to x-axis
pyplot.ylabel('Sin(x)', fontsize = 24) #adding label to y-axis
pyplot.title('Sine Wave', fontsize = 36) #adding the title
pyplot.show() #you can only see plot if you use this statement
```

Histogram

```
import numpy as np  
import matplotlib.pyplot as plt  
  
hBins = np.arange(150, 180, 0.5)
```

Numpy.arange works in same way as range() but the arguments can be floating point as well in addition to integers.

Histogram

```
import numpy as np
import matplotlib.pyplot as plt

hBins = np.arange(150, 180, 0.5)

htI = np.random.randint(157, 163, size=1000)
plt.subplot(211)
histD = plt.hist(htI, label = 'Indian Height', bins=hBins, rwidth = 0.7)
leg = plt.legend(frameon = False)
plt.title('Heights of Indians and Dutchmen (cm)')
plt.grid(True)

htD = np.random.randint(168, 173, size=1000)
plt.subplot(212)
histD = plt.hist(htD, label = 'Dutch Height', bins=hBins, rwidth = 0.7)
leg = plt.legend(frameon = False)

plt.grid(True)

plt.show()
```

Histogram

```
import numpy as np
import matplotlib.pyplot as plt

hBins = np.arange(150, 180, 0.5)

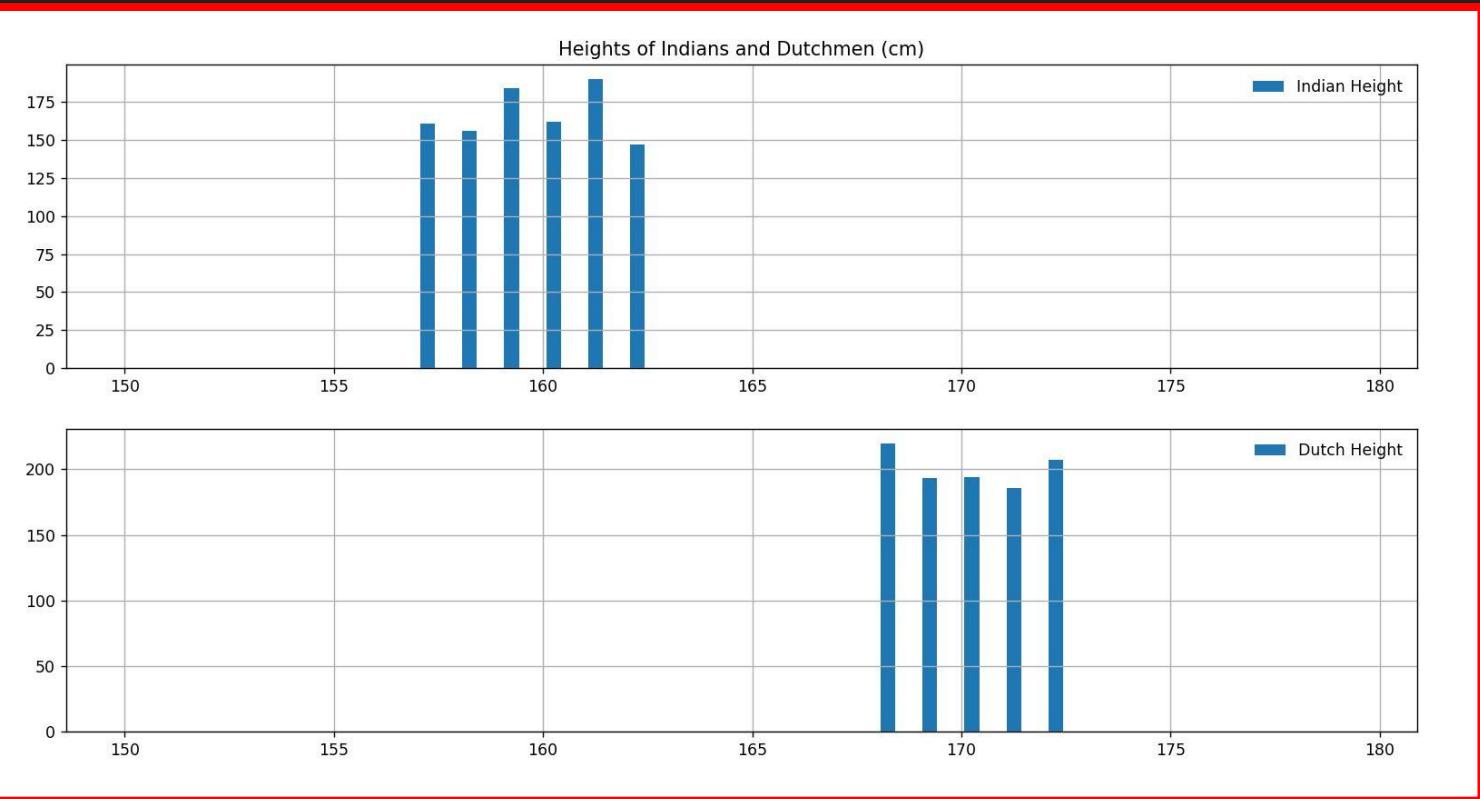
htI = np.random.randint(157, 163, size=1000)
plt.subplot(211)
histD = plt.hist(htI, label = 'Indian Height', bins=hBins, rwidth = 0.7)
leg = plt.legend(frameon = False)
plt.title('Heights of Indians and Dutchmen (cm)')
plt.grid(True)

htD = np.random.randint(168, 173, size=1000)
plt.subplot(212)
histD = plt.hist(htD, label = 'Dutch Height', bins=hBins, rwidth = 0.7)
leg = plt.legend(frameon = False)

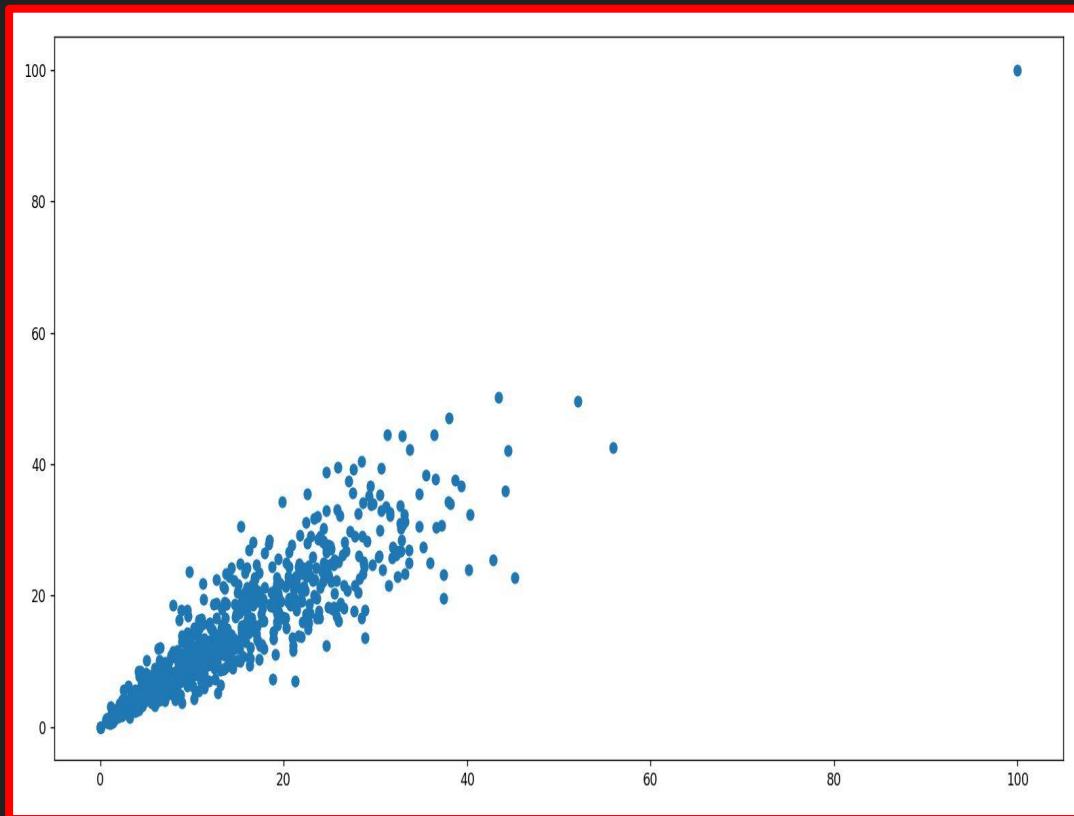
plt.grid(True)

plt.show() #you can only see plot if you use this statement
```

Histogram



Scatter Plot



Scatter Plot

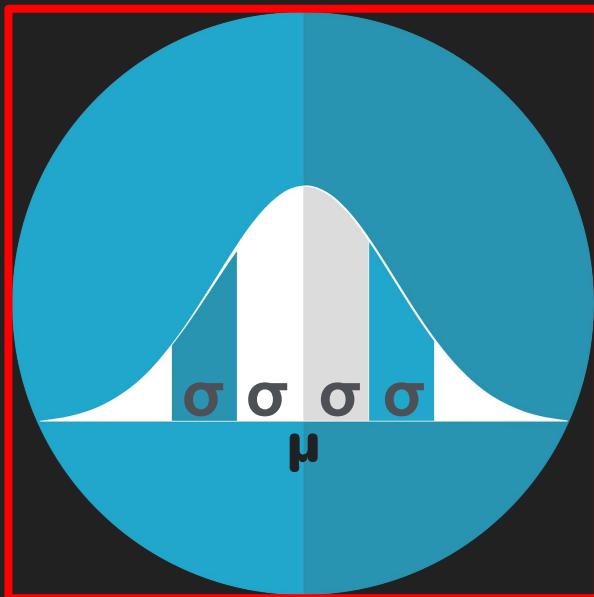
```
import matplotlib.pyplot as plt
import numpy

xValues, yValues = [], []
scatteredness = 0.2

for i in range(40):
    for j in range(40-i):
        xValues.append(i + (i)*numpy.random.normal(0, scatteredness))
        yValues.append(i + (i)*numpy.random.normal(0, scatteredness))
xValues.append(100)
yValues.append(100)

plt.scatter(xValues, yValues)
plt.xlim(0,40)
plt.ylim(0,40)
plt.xlabel('Random x values', fontsize = 24)
plt.ylabel('Random y values', fontsize = 24)
plt.title('Increasingly random gaussian noise added to y and x in y = x', \
          fontsize = 35)
plt.show() #you can only see plot if you use this statement
```

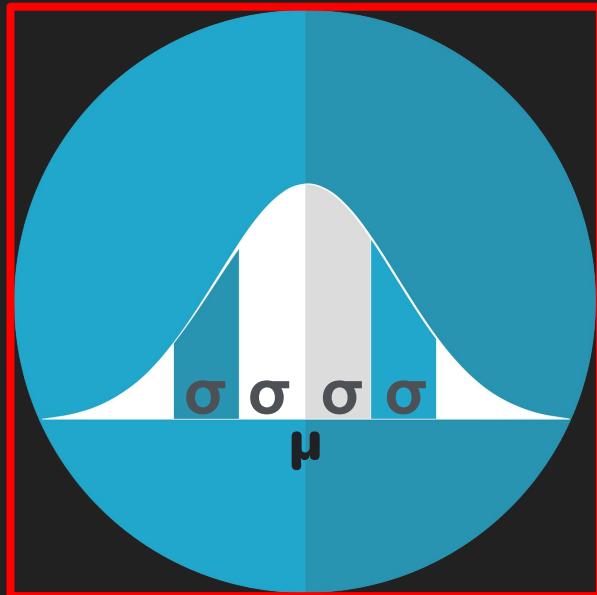
Gaussian Noise



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

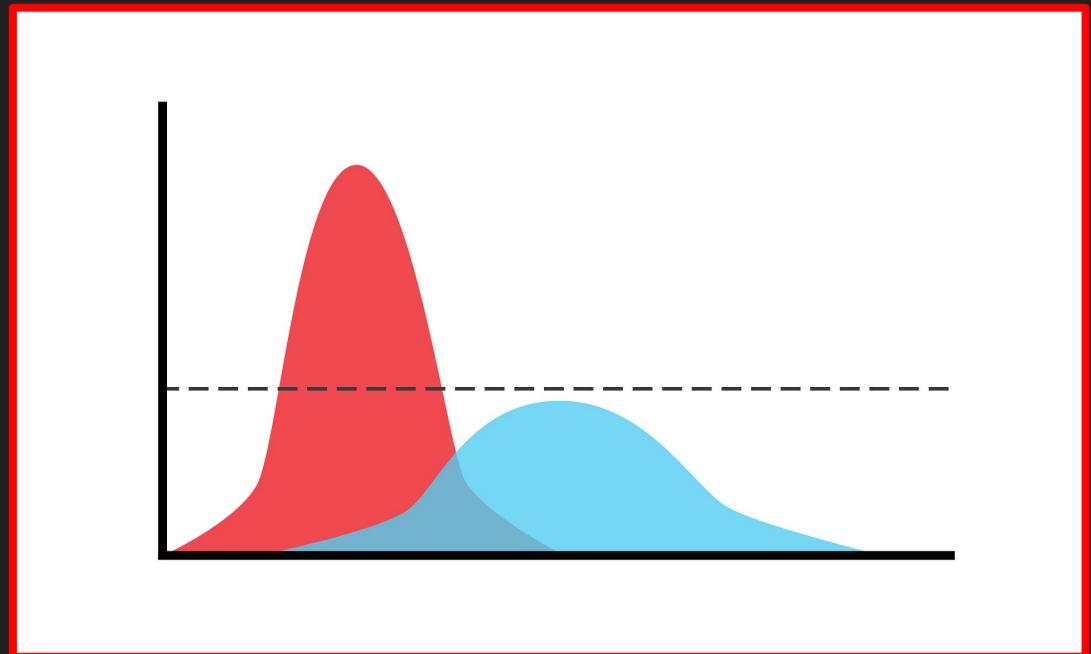
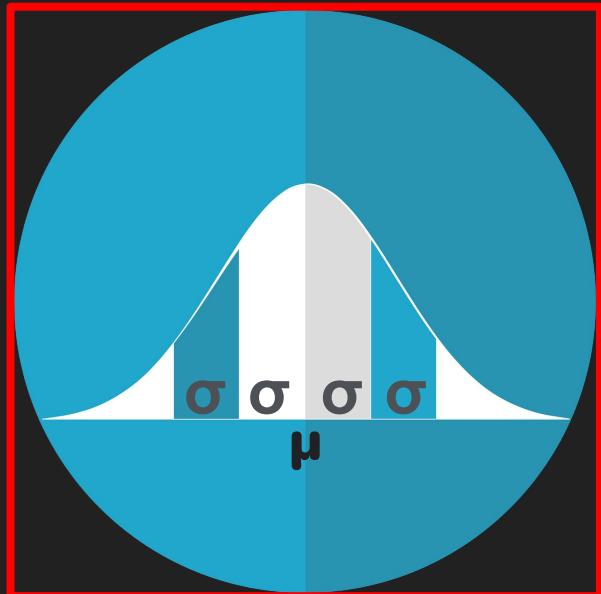
Gaussian Noise

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$



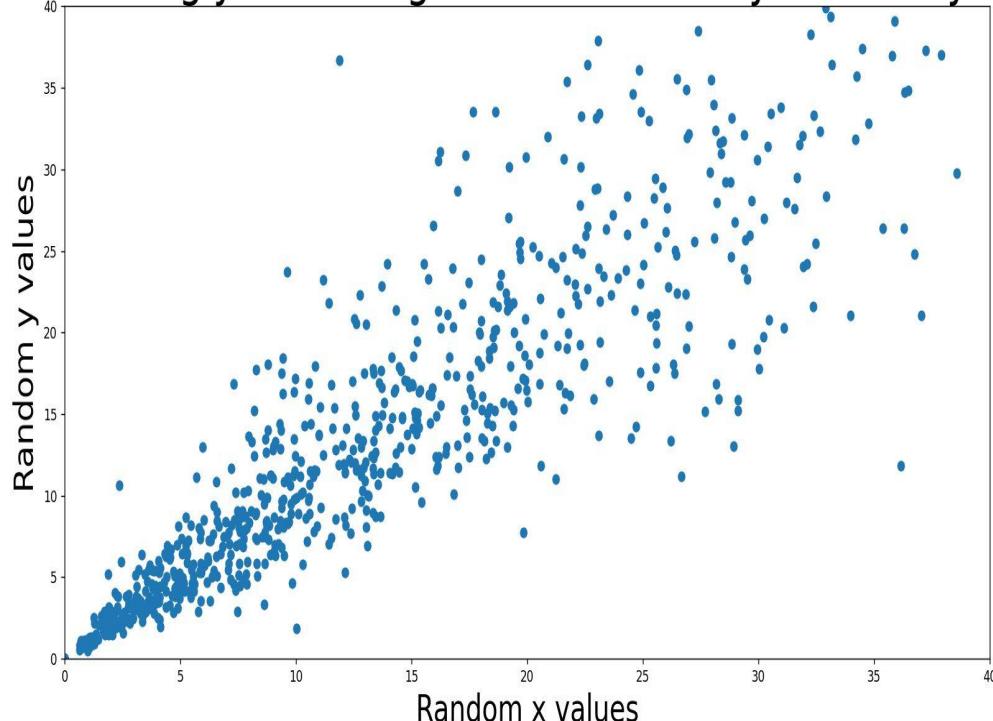
Gaussian Noise

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$



Scatter Plot

Increasingly random gaussian noise to y and x in $y = x$



Scatter Plots

```
import matplotlib.pyplot as plt
import numpy

xValues, yValues = [], []
scatteredness = 0.2

for i in range(40):
    for j in range(40-i):
        xValues.append(i + (i)*numpy.random.normal(0, scatteredness))
        yValues.append(i + (i)*numpy.random.normal(0, scatteredness))
xValues.append(100)
yValues.append(100)

plt.scatter(xValues, yValues)
plt.xlim(0,40)
plt.ylim(0,40)
plt.xlabel('Random x values', fontsize = 24)
plt.ylabel('Random y values', fontsize = 24)
plt.title('Increasingly random gaussian noise added to y and x in y = x', \
          fontsize = 35)
plt.show() #you can only see plot if you use this statement
```

Scatter Plots

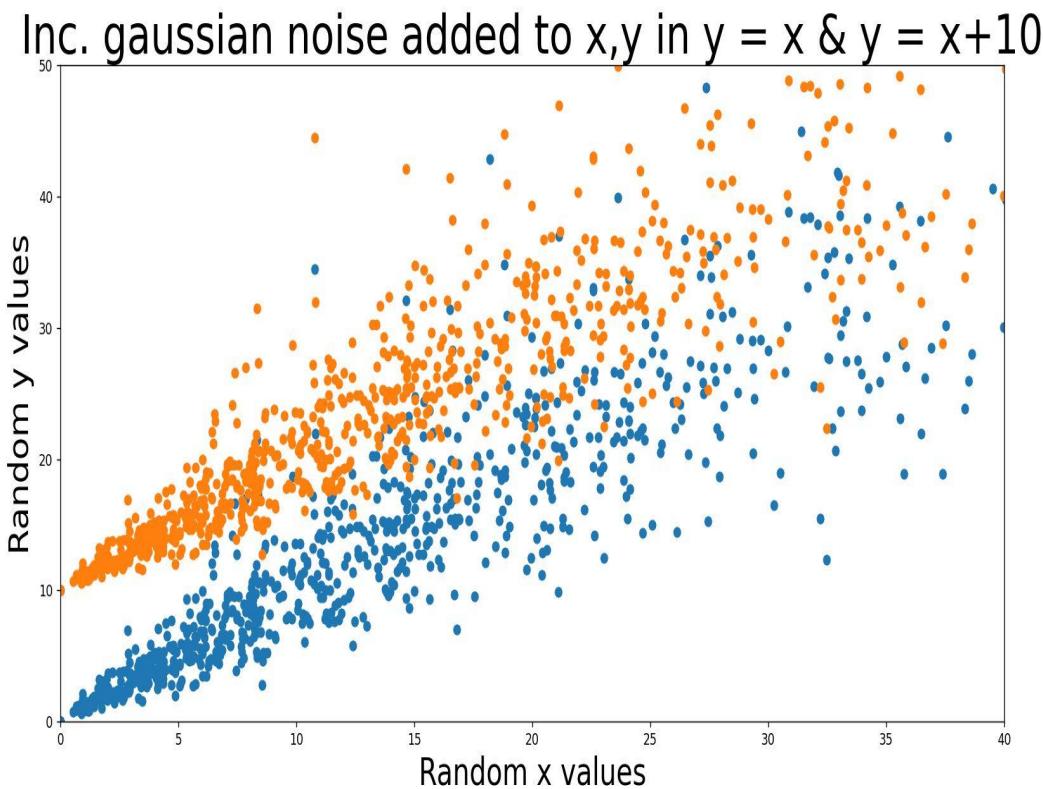
```
import matplotlib.pyplot as plt
import numpy

xValues, yValues = [], []
scatteredness = 0.2

for i in range(40):
    for j in range(40-i):
        xValues.append(i + (i)*numpy.random.normal(0, scatteredness))
        yValues.append(i + (i)*numpy.random.normal(0, scatteredness))
xValues.append(100)
yValues.append(100)

yValues1 = numpy.array(yValues) + 10
plt.scatter(xValues, yValues)
plt.scatter(xValues, yValues1)
plt.xlim(0,40)
plt.ylim(0,50) #40
plt.xlabel('Random x values', fontsize = 24)
plt.ylabel('Random y values', fontsize = 24)
plt.title('Increasingly gaussian noise added to y & x in y = x |& y = x+10', \
          fontsize = 35)
```

Scatter Plot



Announcements

- Surprise quiz before mid-sem
 - No evaluation
 - May give some idea of type of questions in mid-sem
- No class on 9 Dec'22 and 10 Dec'22 (Friday and Saturday)
- From Dec. Lab timings is 3-5 p.m.
- Monday batch students, first 70 ids have lab today (5:30 - 7:30 p.m. @A10,A11 PC labs, and A11-1A)
 - Remaining have lab tomorrow(3-5 p.m.)
- Thursday batch students have lab today (5:30 - 7:30 p.m.)
 - Those having chemistry labs have lab tomorrow (3-5 p.m.)
- From 1-8 Dec'22, get your laptops (if you have/can)
- No Lab on 12 Dec'22

Introducing Numpy

```
x2 = np.random.randint(10, size = (4, 4))
print(x2.shape) #(4, 4)
print(x2)
...
[[9 3 5 2]
 [4 7 6 8]
 [8 1 6 7]
 [7 8 1 5]]
...
```

Using Numpy

```
# importing numpy library  
import numpy as np
```

Using Numpy

```
# importing numpy library
import numpy as np

np.random.seed(0) #seed for reproducibility
```

Using Numpy

```
# importing numpy library
import numpy as np

np.random.seed(0) #seed for reproducibility

x1 = np.random.randint(10, size = 5)
```

Using Numpy

```
# importing numpy library
import numpy as np

np.random.seed(0) #seed for reproducibility

x1 = np.random.randint(10, size = 5)
print(x1.shape) #(5,)
```

Using Numpy

```
# importing numpy library
import numpy as np

np.random.seed(0) #seed for reproducibility

x1 = np.random.randint(10, size = 5)
print(x1.shape) #(5,)
print(x1)
```

Using Numpy

```
# importing numpy library
import numpy as np

np.random.seed(0) #seed for reproducibility

x1 = np.random.randint(10, size = 5)
print(x1.shape) #(5,)
print(x1)
#[5 0 3 3 7]
```

Using Numpy

```
# importing numpy library  
import numpy as np
```

Using Numpy

```
# importing numpy library
import numpy as np

x2 = np.random.randint(10, size = (4, 4))
print(x2.shape) #(4, 4)
print(x2)
```

Using Numpy

```
# importing numpy library
import numpy as np

x2 = np.random.randint(10, size = (4, 4))
print(x2.shape) #(4, 4)
print(x2)
'''
[[9 3 5 2]
 [4 7 6 8]
 [8 1 6 7]
 [7 8 1 5]]
```

Using Numpy

```
# importing numpy library  
import numpy as np
```

```
x3 = np.random.randint(10, size = (5, 4, 3))  
print(x3.shape) #(5, 4, 3)  
print(x3)
```

Using Numpy

```
# importing numpy library
import numpy as np
```

```
x3 = np.random.randint(10, size = (5,4,3))
print(x3.shape) #(5, 4, 3)
print(x3)
```

```
[[[9 8 9]
  [4 3 0]
  [3 5 0]
  [2 3 8]]]
```

```
[[1 3 3]
 [3 7 0]
 [1 9 9]
 [0 4 7]]]
```

```
[[3 2 7]
 [2 0 0]
 [4 5 5]
 [6 8 4]]]
```

```
[[1 4 9]
 [8 1 1]
 [7 9 9]
 [3 6 7]]]
```

```
[[2 0 3]
 [5 9 4]
 [4 6 4]
 [4 3 4]]]
```

Using Numpy: Identity

```
>>> import numpy as np  
>>> myList = [[1,2,3], [4,5,6]]  
>>> mynpArray = np.array(myList, np.int32)
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
>>> print(mynpArray.ndim)
2
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
>>> print(mynpArray.ndim)
2
>>> print(mynpArray.dtype)
int32
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
>>> print(mynpArray.ndim)
2
>>> print(mynpArray.dtype)
int32
>>> print(mynpArray[1,2])
6
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
>>> print(mynpArray.ndim)
2
>>> print(mynpArray.dtype)
int32
>>> print(mynpArray[1,2])
6
>>> myIdentity = np.eye(3)
>>> print(myIdentity)
```

Using Numpy: Identity

```
>>> import numpy as np
>>> myList = [[1,2,3], [4,5,6]]
>>> mynpArray = np.array(myList, np.int32)
>>> print(type(myList), type(mynpArray))
<class 'list'> <class 'numpy.ndarray'>
>>> print(mynpArray)
[[1 2 3]
 [4 5 6]]
>>> print(mynpArray.ndim)
2
>>> print(mynpArray.dtype)
int32
>>> print(mynpArray[1,2])
6
>>> myIdentity = np.eye(3)
>>> print(myIdentity)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Using Numpy: Ones

```
>>> print(np.ones((2,2,3)))
```

Using Numpy: Ones

```
>>> print(np.ones((2,2,3)))  
[[[1. 1. 1.]  
 [1. 1. 1.]]  
  
 [[1. 1. 1.]  
 [1. 1. 1.]]]
```

Using Numpy: Ones

```
>>> print(np.ones((2,2,3)))
[[[1. 1. 1.]
 [1. 1. 1.]]]

[[1. 1. 1.]
 [1. 1. 1.]]]
```

```
>>> print(np.ones(2,2,3))
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    print(np.ones(2,2,3))
  File "C:\Users\Rohit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\numpy\core\numeric.py", line 204, in ones
    a = empty(shape, dtype, order)
TypeError: Cannot interpret '2' as a data type
>>> print(np.ones(2,2,3))
KeyboardInterrupt
```

Using Numpy: Zeros

```
>>> np.zeros( (2, 2) )
array( [ [ 0.,  0. ],
        [ 0.,  0. ] ] )
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
>>> myArray[4:7]
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
>>> myArray[4:7]
array([4, 5, 6])
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
>>> myArray[4:7]
array([4, 5, 6])
>>> myArray[::-2]
array([0, 2, 4, 6, 8])
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
>>> myArray[4:7]
array([4, 5, 6])
>>> myArray[::-2]
array([0, 2, 4, 6, 8])
>>> myArray[::-1]
```

Using Numpy: Indexing/Slicing

```
>>> myArray = np.arange(10)
>>> myArray[:4]
array([0, 1, 2, 3])
>>> myArray[4:7]
array([4, 5, 6])
>>> myArray[::-2]
array([0, 2, 4, 6, 8])
>>> myArray[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Using Numpy: 2D Slicing

```
>>> my2DArray = np.random.randint(20, size = (3,4))
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 17],
       [ 5,  9,  3,  0]])
>>> my2DArray[1,3] = 100
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 100],
       [ 5,  9,  3,  0]])
>>> my2DArray[:2,:3]
```

Using Numpy: 2D Slicing

```
>>> my2DArray = np.random.randint(20, size = (3,4))
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 17],
       [ 5,  9,  3,  0]])
>>> my2DArray[1,3] = 100
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 100],
       [ 5,  9,  3,  0]])
>>> my2DArray[:2,:3]
array([[ 8, 14, 15],
       [15, 13, 16]])
>>> my2DArray[:, ::2]
```

Using Numpy: 2D Slicing

```
>>> my2DArray = np.random.randint(20, size = (3,4))
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 17],
       [ 5,  9,  3,  0]])
>>> my2DArray[1,3] = 100
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 100],
       [ 5,  9,  3,  0]])
>>> my2DArray[:2,:3]
array([[ 8, 14, 15],
       [15, 13, 16]])
>>> my2DArray[:, ::2]
array([[ 8, 15],
       [15, 16],
       [ 5,  3]])
```

Using Numpy: SubArrays and No Copy Views

```
>>> my2DArray
array([[ 8, 14, 15,  3],
       [15, 13, 16, 17],
       [ 5,   9,   3,  0]])
```

```
>>> my2DArraySub = my2DArray[:2, :2]
>>> my2DArraySub[0, 0] = 999
>>> my2DArraySub
array([[999, 14],
       [15, 13]])
>>> my2DArray
```

Using Numpy: SubArrays and No Copy Views

```
>>> my2DArray
array([[ 8, 14, 15, 3],
       [15, 13, 16, 17],
       [ 5,  9,  3,  0]])
```

```
>>> my2DArraySub = my2DArray[:2, :2]
>>> my2DArraySub[0, 0] = 999
>>> my2DArraySub
array([[999, 14],
       [15, 13]])
>>> my2DArray
array([[999, 14, 15, 3],
       [15, 13, 16, 100],
       [ 5,  9,  3,  0]])
```

Making Copies

```
>>> my4Darray = np.random.randint(0, 50, (1,2,3,4))  
>>> my4Darray  
array([[[[29,  1, 31, 44],  
        [24, 24,  3, 18],  
        [47,  3, 42, 12]],  
  
       [[38, 35, 22,  5],  
        [23, 43, 32, 11],  
        [40, 20, 10, 43]]])
```

Making Copies

```
>>> my4Darray  
array([[[[29, 1, 31, 44],  
        [24, 24, 3, 18],  
        [47, 3, 42, 12]],  
  
       [[38, 35, 22, 5],  
        [23, 43, 32, 11],  
        [40, 20, 10, 43]]])
```

Making Copies

```
>>> my4DsubArray = my4Darray[:, :, :2, :2].copy()  
>>> my4DsubArray  
array([[[[29,  1,  31,  44],  
        [24,  24,  3,  18],  
        [47,  3,  42,  12]],  
  
       [[38,  35,  22,  5],  
        [23,  43,  32,  11],  
        [40,  20,  10,  43]]]])
```

```
>>> my4Darray  
array([[[[29,  1,  31,  44],  
        [24,  24,  3,  18],  
        [47,  3,  42,  12]],  
  
       [[38,  35,  22,  5],  
        [23,  43,  32,  11],  
        [40,  20,  10,  43]]]])
```

Making Copies

```
>>> my4DsubArray = my4Darray[:, :, :2, :2].copy()  
>>> my4DsubArray  
array([[[[29, 1, 31, 44],  
        [24, 24, 3, 18],  
        [47, 3, 42, 12]],  
  
       [[38, 35, 22, 5],  
        [23, 43, 32, 11],  
        [40, 20, 10, 43]]])  
  
>>> my4DsubArray[0,0,0,0] = 1000  
>>> my4DsubArray  
array([[[[1000, 1,  
        [ 24, 24]],  
  
       [[ 38, 35],  
        [ 23, 43]]]])
```

```
>>> my4Darray  
array([[[[29, 1, 31, 44],  
        [24, 24, 3, 18],  
        [47, 3, 42, 12]],  
  
       [[38, 35, 22, 5],  
        [23, 43, 32, 11],  
        [40, 20, 10, 43]]]])
```

Making Copies

```
>>> my4DsubArray = my4Darray[:, :, :2, :2].copy()
>>> my4DsubArray
array([[[[29,  1,  31,  44],
         [24,  24,  3,  18],
         [47,  3,  42,  12]],

        [[38,  35,  22,  5],
         [23,  43,  32,  11],
         [40,  20,  10,  43]]]])
```



```
>>> my4DsubArray[0,0,0,0] = 1000
>>> my4DsubArray
array([[[[1000,      1,
           [ 24,      24]],

          [[ 38,      35],
           [ 23,      43]]]])
```



```
>>> my4Darray
```

```
>>> my4Darray
array([[[[29,  1,  31,  44],
         [24,  24,  3,  18],
         [47,  3,  42,  12]],

        [[38,  35,  22,  5],
         [23,  43,  32,  11],
         [40,  20,  10,  43]]]])
```

Making Copies

```
>>> my4DsubArray = my4Darray[:, :, :2, :2].copy()
>>> my4DsubArray
array([[[[29,  1, 31, 44],
         [24, 24,  3, 18],
         [47,  3, 42, 12]],

        [[38, 35, 22,  5],
         [23, 43, 32, 11],
         [40, 20, 10, 43]]]])
```



```
>>> my4DsubArray[0,0,0,0] = 1000
>>> my4DsubArray
array([[[[1000,      1],
          [ 24,      24]],

         [[ 38,      35],
          [ 23,      43]]]])
```



```
>>> my4Darray
array([[[[29,  1, 31, 44],
         [24, 24,  3, 18],
         [47,  3, 42, 12]],

        [[38, 35, 22,  5],
         [23, 43, 32, 11],
         [40, 20, 10, 43]]]])
```

```
>>> my4Darray
array([[[[29,  1, 31, 44],
         [24, 24,  3, 18],
         [47,  3, 42, 12]],

        [[38, 35, 22,  5],
         [23, 43, 32, 11],
         [40, 20, 10, 43]]]])
```

Reshape

```
# importing numpy library  
import numpy as np
```

```
>>> npArray = np.arange(0.1,1,0.1)  
>>> grid = npArray.reshape((3,3))  
>>> npArray
```

Reshape

```
# importing numpy library
import numpy as np
```

```
>>> npArray = np.arange(0.1,1,0.1)
>>> grid = npArray.reshape((3,3))
>>> npArray
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>>> grid
```

Reshape

```
# importing numpy library
import numpy as np
```

```
>>> npArray = np.arange(0.1,1,0.1)
>>> grid = npArray.reshape((3,3))
>>> npArray
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>>> grid
array([[0.1, 0.2, 0.3],
       [0.4, 0.5, 0.6],
       [0.7, 0.8, 0.9]])
```

Reshape

```
# importing numpy library
import numpy as np
```

```
>>> npArray = np.arange(0.1,1,0.1)
>>> grid = npArray.reshape((3,3))
>>> npArray
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>>> grid
array([[0.1, 0.2, 0.3],
       [0.4, 0.5, 0.6],
       [0.7, 0.8, 0.9]])
>>> grid = np.arange(0.1,1.1,0.1).reshape((3,3))
```

Reshape

```
# importing numpy library
import numpy as np
```

```
>>> npArray = np.arange(0.1,1,0.1)
>>> grid = npArray.reshape((3,3))
>>> npArray
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
>>> grid
array([[0.1, 0.2, 0.3],
       [0.4, 0.5, 0.6],
       [0.7, 0.8, 0.9]])
>>> grid = np.arange(0.1,1.1,0.1).reshape((3,3))
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    grid = np.arange(0.1,1.1,0.1).reshape((3,3))
ValueError: cannot reshape array of size 10 into shape (3,3)
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])  
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([99, 99])  
>>> array3 = np.array([3,2,1])  
>>> arrayCat = np.concatenate([array1, array2, array3])  
>>> arrayCat
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])  
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([99, 99])  
>>> array3 = np.array([3,2,1])  
>>> arrayCat = np.concatenate([array1, array2, array3])  
>>> arrayCat  
array([ 1,  2,  3, 99, 99,  3,  2,  1])
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])  
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([99, 99])  
>>> array3 = np.array([3,2,1])  
>>> arrayCat = np.concatenate([array1, array2, array3])  
>>> arrayCat  
array([ 1,  2,  3, 99, 99,  3,  2,  1])  
>>> arr1, arr2, arr3 = np.split(arrayCat, [3,6])
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])  
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([99, 99])  
>>> array3 = np.array([3,2,1])  
>>> arrayCat = np.concatenate([array1, array2, array3])  
>>> arrayCat  
array([ 1,  2,  3, 99, 99,  3,  2,  1])  
>>> arr1, arr2, arr3 = np.split(arrayCat, [3,6])  
>>> print(arr1, arr2, arr3)
```

Concatenate & Split

```
# importing numpy library  
import numpy as np
```

```
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([1,2,3])  
>>> arrayCat = np.concatenate([array1, array2])  
>>> arrayCat  
array([1, 2, 3, 1, 2, 3])  
>>> array1 = np.array([1,2,3])  
>>> array2 = np.array([99, 99])  
>>> array3 = np.array([3,2,1])  
>>> arrayCat = np.concatenate([array1, array2, array3])  
>>> arrayCat  
array([ 1,  2,  3, 99, 99,  3,  2,  1])  
>>> arr1, arr2, arr3 = np.split(arrayCat, [3,6])  
>>> print(arr1, arr2, arr3)  
[1 2 3] [99 99 3] [2 1]
```

Numpy Universal Functions (ufunc)

```
>>> arr1 = np.array([1, 2, 3, 4])
>>> arr2 = np.array([4, 3, 2, 1])
>>> arrSum = np.add(arr1, arr2)
```

Numpy Universal Functions (ufunc)

```
>>> arr1 = np.array([1, 2, 3, 4])
>>> arr2 = np.array([4, 3, 2, 1])
>>> arrSum = np.add(arr1, arr2)
>>> arrSum
array([5, 5, 5, 5])
```

Vectorized Operations to Speed Up Loops

```
import numpy as np
import time

np.random.seed(0) #for reproducibility

def computeReciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1/values[i]
    return output
```

Vectorized Operations to Speed Up Loops

```
import numpy as np
import time

np.random.seed(0) #for reproducibility

def computeReciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1/values[i]
    return output

values = np.random.randint(1, 10, size = 10**7)
```

Vectorized Operations to Speed Up Loops

```
import numpy as np
import time

np.random.seed(0) #for reproducibility

def computeReciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1/values[i]
    return output

values = np.random.randint(1, 10, size = 10**7)

tic = time.perf_counter()
newValuesLoop = computeReciprocals(values)
print(newValuesLoop[:3], newValuesLoop[-3:])
toc = time.perf_counter()
print(f'Time taken using loop: {toc - tic:0.4f} seconds')
```

Vectorized Operations to Speed Up Loops

```
import numpy as np
import time

np.random.seed(0) #for reproducibility

def computeReciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1/values[i]
    return output

values = np.random.randint(1, 10, size = 10**7)

tic = time.perf_counter()
newValuesLoop = computeReciprocals(values)
print(newValuesLoop[:3], newValuesLoop[-3:])
toc = time.perf_counter()
print(f'Time taken using loop: {toc - tic:0.4f} seconds')

tic = time.perf_counter()
newValuesVectorized = 1/values
print(newValuesVectorized[:3], newValuesVectorized[-3:])
toc = time.perf_counter()
print(f'Time taken using universal function (ufunc): {toc - tic:0.4f} seconds')
```

Vectorized Operations to Speed Up Loops

```
import numpy as np
import time

np.random.seed(0) #for reproducibility

def computeReciprocals(values):
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1/values[i]
    return output

values = np.random.randint(1, 10, size = 10**7)

tic = time.perf_counter()
newValuesLoop = computeReciprocals(values)
print(newValuesLoop[:3], newValuesLoop[-3:])
toc = time.perf_counter()
```

```
[0.16666667 1.          0.25         ] [0.125 0.5      0.5      ]
Time taken using loop: 2.0691 seconds
[0.16666667 1.          0.25         ] [0.125 0.5      0.5      ]
Time taken using universal function (ufunc): 0.0384 seconds
```

Vectorized Operations to Speed Up Loops

```
>>> np.arange(5)/np.arange(1, 6)
```

Vectorized Operations to Speed Up Loops

```
>>> np.arange(5)/np.arange(1,6)
array([0.          , 0.5         , 0.66666667, 0.75         , 0.8        ])
```

Vectorized Operations to Speed Up Loops

```
>>> np.arange(5)/np.arange(1, 6)
array([0.          , 0.5         , 0.66666667, 0.75         ,
       0.8         ])
>>> 2**np.arange(9).reshape((3, 3))
```

Vectorized Operations to Speed Up Loops

```
>>> np.arange(5)/np.arange(1,6)
array([0.          , 0.5         , 0.66666667, 0.75         , 0.8        ])
>>> 2**np.arange(9).reshape((3,3))
array([[ 1,   2,   4],
       [ 8,  16,  32],
       [ 64, 128, 256]], dtype=int32)
```

Aggregation Functions

```
>>> import numpy as np
>>> bigNpArray = np.random.rand(10**6) # uniform distribution over [0, 1)
>>> print(min(bigNpArray), max(bigNpArray)) # python built-in
1.7216838649192212e-06 0.999997536256589
```

Aggregation Functions

```
>>> import numpy as np
>>> bigNpArray = np.random.rand(10**6) # uniform distribution over [0, 1)
>>> print(min(bigNpArray), max(bigNpArray)) # python built-in
1.7216838649192212e-06 0.999997536256589
>>> print(np.min(bigNpArray), np.max(bigNpArray)) # numpy built-in
1.7216838649192212e-06 0.999997536256589
```

Aggregation Functions

```
>>> import numpy as np
>>> bigNpArray = np.random.rand(10**6) # uniform distribution over [0, 1)
>>> print(min(bigNpArray), max(bigNpArray)) # python built-in
1.7216838649192212e-06 0.999997536256589
>>> print(np.min(bigNpArray), np.max(bigNpArray)) # numpy built-in
1.7216838649192212e-06 0.999997536256589
>>> print(bigNpArray.min(), bigNpArray.max()) #ndarray's method
1.7216838649192212e-06 0.999997536256589
```

Random Vs Rand

```
>>> M = np.random.random([3 ,4])
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
```

Random Vs Rand

```
>>> M = np.random.random([3 ,4])
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> np.random.seed(0)
>>> M = np.random.random((3 ,4))
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
```

Random Vs Rand

```
>>> M = np.random.random([3 ,4])
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> np.random.seed(0)
>>> M = np.random.random((3 ,4))
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> np.random.seed(0)
>>> M = np.random.rand((3 ,4))
```

Random Vs Rand

```
>>> M = np.random.random([3 ,4])
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> np.random.seed(0)
>>> M = np.random.random((3 ,4))
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> np.random.seed(0)
>>> M = np.random.rand((3 ,4))
```

Random Vs Rand

```
>>> np.random.seed(0)
>>> M = np.random.rand((3 ,4))
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    M = np.random.rand((3 ,4))
  File "mtrand.pyx", line 1182, in numpy.random.mtrand.RandomState.rand
  File "mtrand.pyx", line 425, in numpy.random.mtrand.RandomState.random_sample
  File "_common.pyx", line 307, in numpy.random._common.double_fill
TypeError: 'tuple' object cannot be interpreted as an integer
>>> M = np.random.rand([3 ,4])
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    M = np.random.rand([3 ,4])
  File "mtrand.pyx", line 1182, in numpy.random.mtrand.RandomState.rand
  File "mtrand.pyx", line 425, in numpy.random.mtrand.RandomState.random_sample
  File "_common.pyx", line 307, in numpy.random._common.double_fill
TypeError: 'list' object cannot be interpreted as an integer
```

```
>>> M = np.random.random([3 ,4])
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
```

```
>>> np.random.seed(0)
>>> M = np.random.random((3 ,4))
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
```

```
>>> np.random.seed(0)
>>> M = np.random.rand((3 ,4))
```

Random Vs Rand

```
>>> np.random.seed(0)
>>> M = np.random.rand((3 ,4))
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    M = np.random.rand((3 ,4))
  File "mtrand.pyx", line 1182, in numpy.random.mtrand.RandomState.rand
  File "mtrand.pyx", line 425, in numpy.random.mtrand.RandomState.random_sample
  File "_common.pyx", line 307, in numpy.random._common.double_fill
TypeError: 'tuple' object cannot be interpreted as an integer
>>> M = np.random.rand([3 ,4])
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    M = np.random.rand([3 ,4])
  File "mtrand.pyx", line 1182, in numpy.random.mtrand.RandomState.rand
  File "mtrand.pyx", line 425, in numpy.random.mtrand.RandomState.random_sample
  File "_common.pyx", line 307, in numpy.random._common.double_fill
TypeError: 'list' object cannot be interpreted as an integer
>>> M = np.random.rand(3 ,4)
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
```

Aggregation Functions

```
>>> M = np.random.rand(3 ,4)
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> M.sum()
7.478282790980994
>>> M.min(axis=0)
```

Aggregation Functions

```
>>> M = np.random.rand(3 ,4)
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> M.sum()
7.478282790980994
>>> M.min(axis=0)
array([0.4236548 , 0.38344152, 0.43758721, 0.52889492])
>>> M.max(axis=1)
```

Aggregation Functions

```
>>> M = np.random.rand(3 ,4)
>>> M
array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
       [0.4236548 , 0.64589411, 0.43758721, 0.891773 ],
       [0.96366276, 0.38344152, 0.79172504, 0.52889492]])
>>> M.sum()
7.478282790980994
>>> M.min(axis=0)
array([0.4236548 , 0.38344152, 0.43758721, 0.52889492])
>>> M.max(axis=1)
array([0.71518937, 0.891773 , 0.96366276])
```

Numpy in Action

$$x_1 - 2x_2 + x_3 = 0$$

$$2x_2 - 8x_3 = 8 \quad \text{system of equations}$$

$$5x_1 - 5x_3 = 10$$

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

Numpy in Action

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

$$x_1 - 2x_2 + x_3 = 0$$

$$2x_2 - 8x_3 = 8$$

$$5x_1 - 5x_3 = 10$$

Numpy in Action

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

```
# Solves Ax = b
import numpy as np
import numpy.linalg as la

A = np.array([[1, -2, 1], [0, 2, -8], [5, 0, -5]])
b = np.array([[0], [8], [10]])

print('Rank of coff. matrix is', la.matrix_rank(A))

print()

sol_inv = np.matmul(la.inv(A), b)
print(sol_inv)
```

$$x_1 - 2x_2 + x_3 = 0$$

$$2x_2 - 8x_3 = 8$$

$$5x_1 - 5x_3 = 10$$

Numpy in Action

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

```
# Solves Ax = b
import numpy as np
import numpy.linalg as la

A = np.array([[1, -2, 1], [0, 2, -8], [5, 0, -5]])
b = np.array([[0], [8], [10]])

print('Rank of coff. matrix is', la.matrix_rank(A))

print()

sol_inv = np.matmul(la.inv(A), b)
print(sol_inv)

print()
sol_la_solve = la.solve(A,b)
print(sol_la_solve)
```

$$x_1 - 2x_2 + x_3 = 0$$

$$2x_2 - 8x_3 = 8$$

$$5x_1 - 5x_3 = 10$$

Numpy in Action

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

```
# Solves Ax = b
import numpy as np
import numpy.linalg as la

A = np.array([[1, -2, 1], [0, 2, -8], [5, 0, -5]])
b = np.array([[0], [8], [10]])

print('Rank of coff. matrix is', np.linalg.matrix_rank(A))
print()

sol_inv = np.matmul(la.inv(A), b)
print(sol_inv)

print()
sol_la_solve = la.solve(A, b)
print(sol_la_solve)
```

x₁ - 2x₂ + x₃ = 0
Rank of coff. matrix is 3
[[1.00000000e+00] = 8
[2.77555756e-16]
[-1.00000000e+00]]
[[1.] = 10
[0.]
[-1.]]

Numpy in Action: Eigenvalues and Eigenvectors

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

Numpy in Action: Eigenvalues and Eigenvectors

```
#calculate eigenvalues and eigenvectors of matrix A  
  
import numpy as np  
import numpy.linalg as la  
  
A = np.array([[1,6],[5,2]])
```

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

Numpy in Action

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 5 & 0 & -5 \end{bmatrix}$$

Coefficient matrix

```
# Solves Ax = b
import numpy as np
import numpy.linalg as la

A = np.array([[1, -2, 1], [0, 2, -8], [5, 0, -5]])
b = np.array([[0], [8], [10]])

print('Rank of coff. matrix is', np.linalg.matrix_rank(A))
print()

sol_inv = np.matmul(la.inv(A), b)
print(sol_inv)

print()
sol_la_solve = la.solve(A, b)
print(sol_la_solve)
```

x₁ - 2x₂ + x₃ = 0
Rank of coff. matrix is 3
[[1.00000000e+00] = 8
[2.77555756e-16]
[-1.00000000e+00]]
[[1.] = 10
[0.]
[-1.]]

Numpy in Action: Eigenvalues and Eigenvectors

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

Numpy in Action: Eigenvalues and Eigenvectors

```
#calculate eigenvalues and eigenvectors of matrix A  
  
import numpy as np  
import numpy.linalg as la  
  
A = np.array([[1,6],[5,2]])
```

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

Numpy in Action: Eigenvalues and Eigenvectors

```
#calculate eigenvalues and eigenvectors of matrix A  
  
import numpy as np  
import numpy.linalg as la  
  
A = np.array([[1,6],[5,2]])  
w,v = la.eig(A)  
print(w)  
print(v)
```

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

Numpy in Action: Eigenvalues and Eigenvectors

```
#calculate eigenvalues and eigenvectors of matrix A  
  
import numpy as np  
import numpy.linalg as la  
  
A = np.array([[1,6],[5,2]])  
w,v = la.eig(A)  
print(w)  
print(v)
```

$$A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$$

```
[-4.  7.]  
[ [-0.76822128 -0.70710678]  
  [ 0.6401844  -0.70710678] ]
```

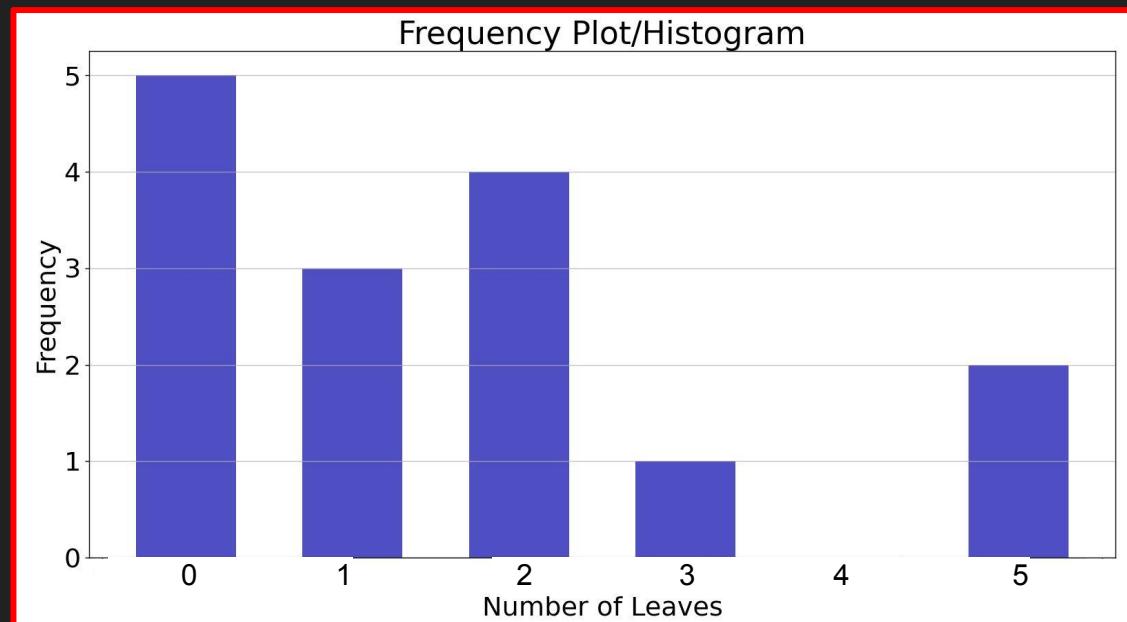
Data Statistics

Data Statistics: Terms

- **Population**: Collection of elements under study.
- E.g. data about COVID over the globe -> set of all people over earth
- **Sample**: Subset of population, practical to study.
- Collect attributes' values, e.g. COVID +ve or -ve
- **Sample point**: each element of sample
- Discrete data: values are from countable finite set. Cannot include negative integers, fractions, or numbers with decimal. E.g. grades, genders.
- Continuous data: values are from uncountable infinite set. E.g. height, speed.
- How to deal with continuous data?
- Use bins or intervals

Describing Datasets: Bar Plots

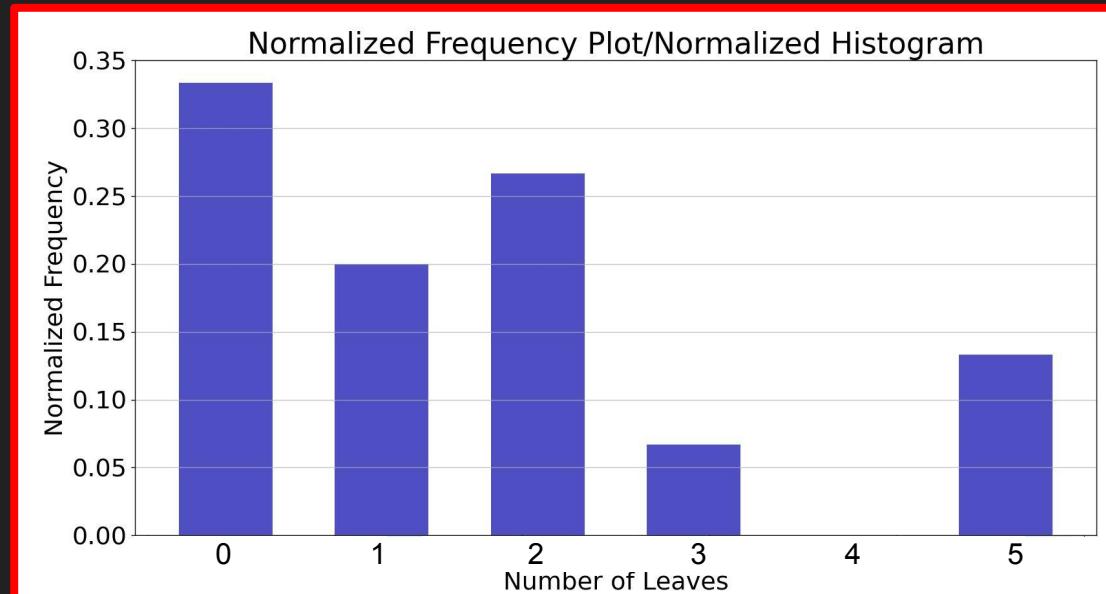
Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15



How will you construct this table and make the plot?
0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 5, 5

Describing Datasets: Bar Plots

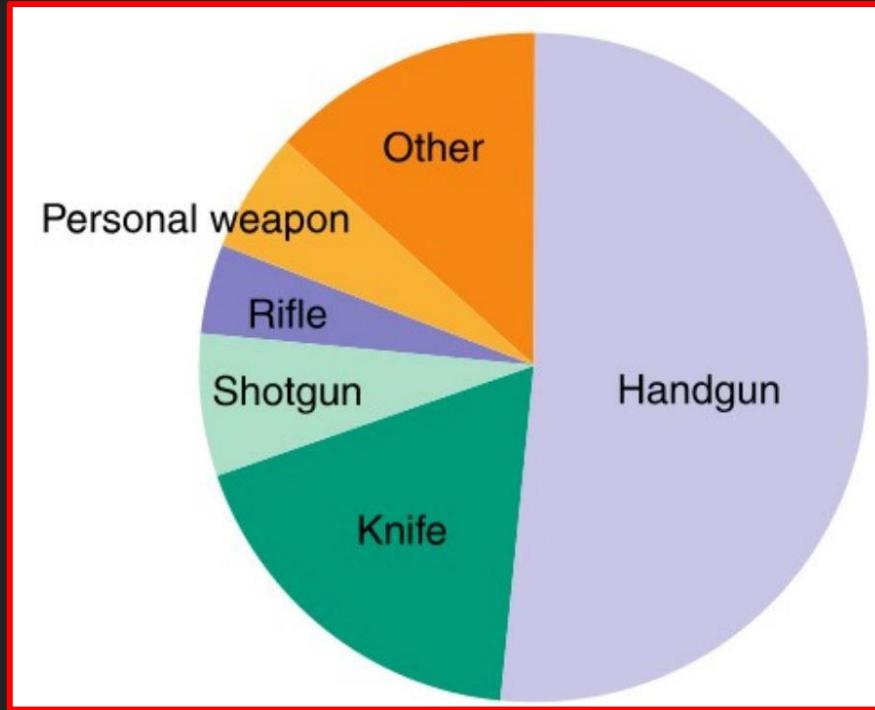
Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15



Sometimes normalized frequency is more convenient. Sum of frequency values = 1.

Describing Datasets: Pie Chart

Pie chart: for non-numerical data Visualization of relative frequency plot



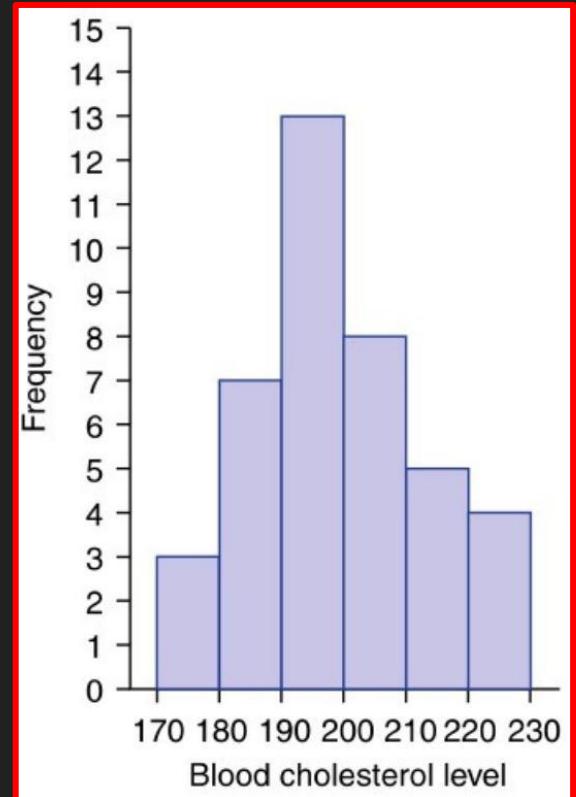
**Weapons used
in crimes.**

Describing Datasets: Histograms

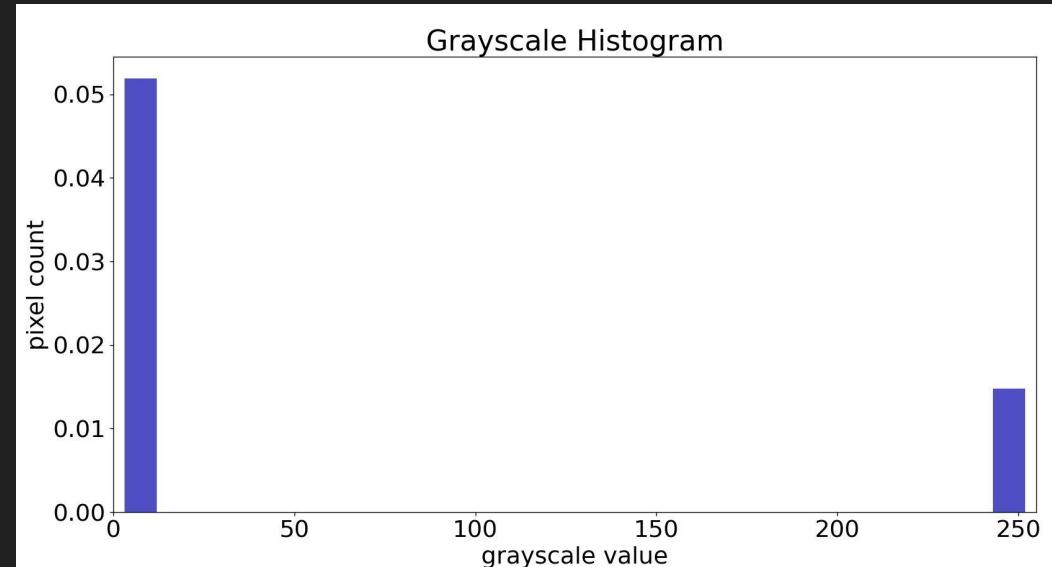
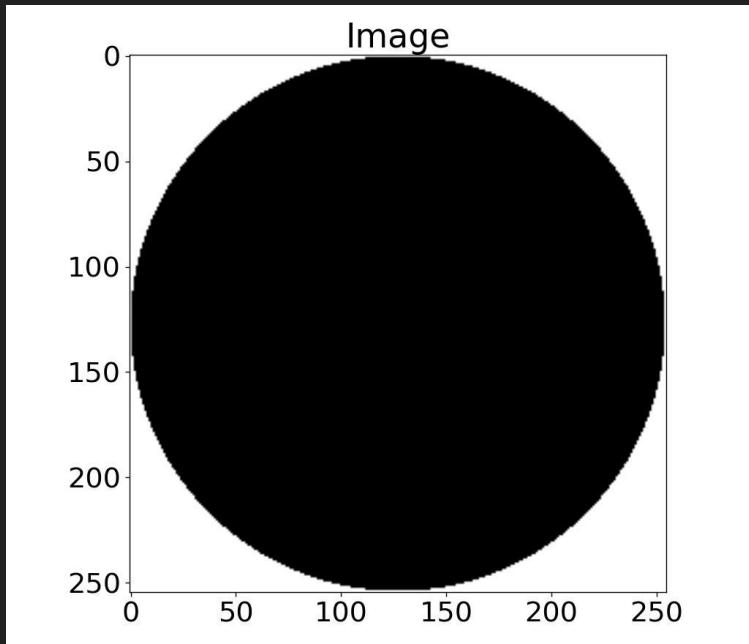
- When unique values are numerous
- Use Bins or Class Intervals

213	174	193	196	220	183	194	200
192	200	200	199	178	183	188	193
187	181	193	205	196	211	202	213
216	206	195	191	171	194	184	191
221	212	221	204	204	191	183	227

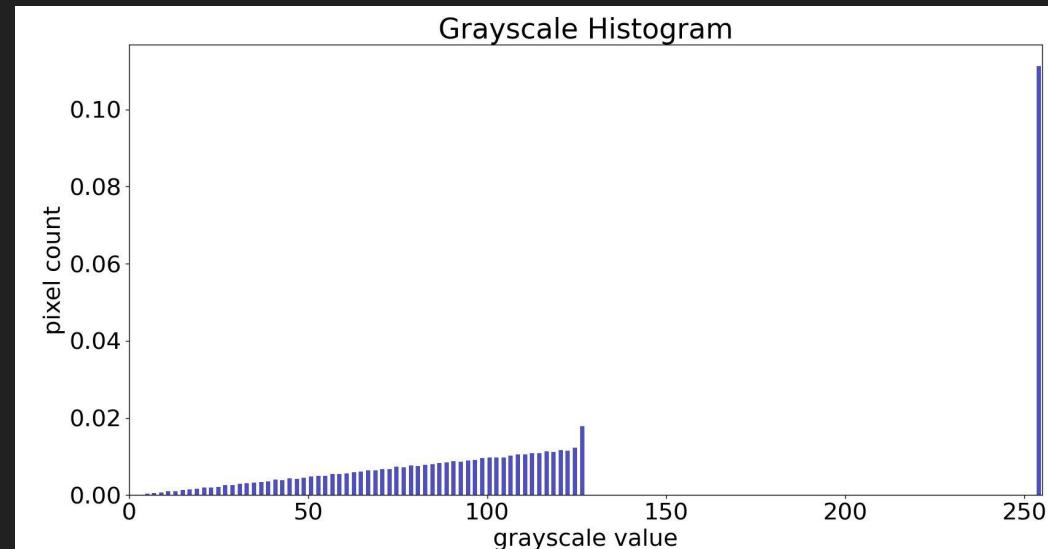
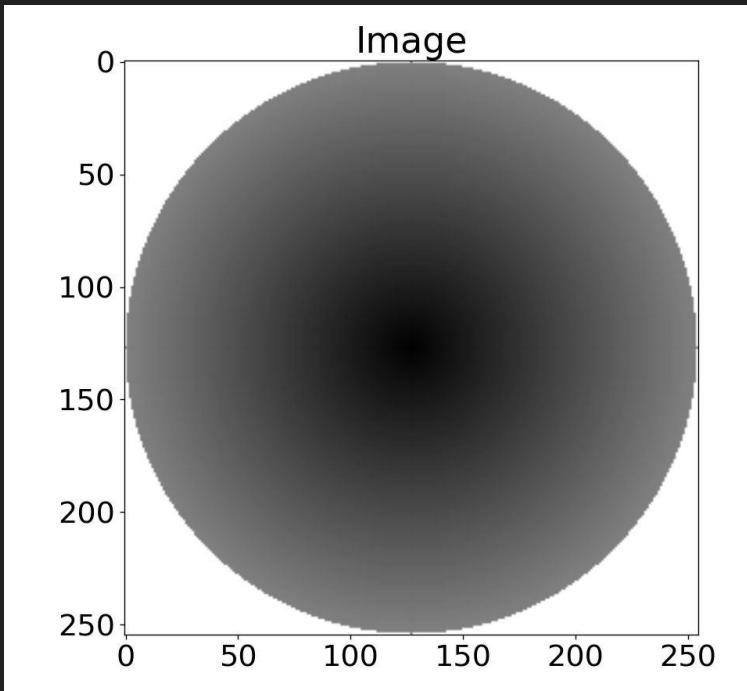
Data: Blood cholesterol levels



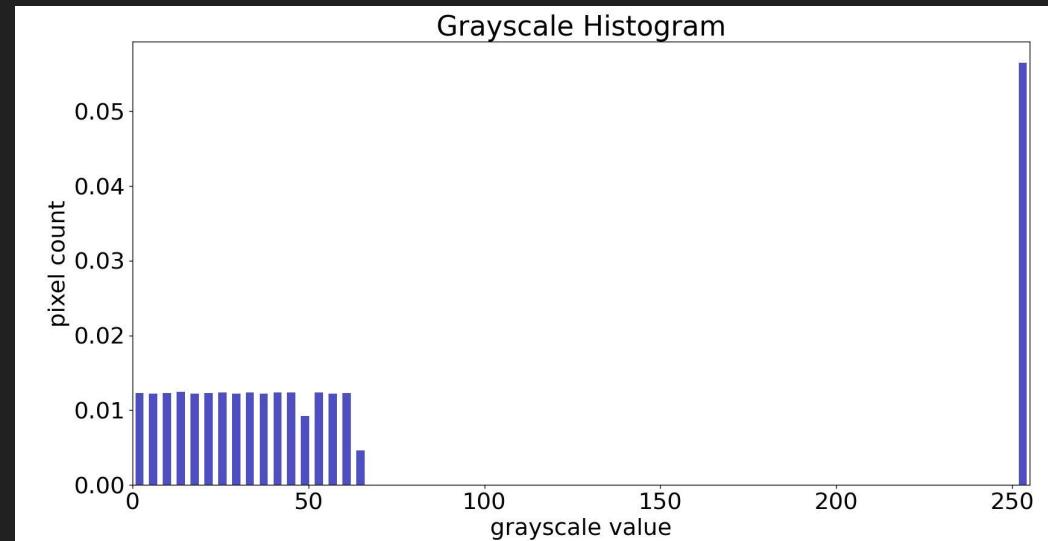
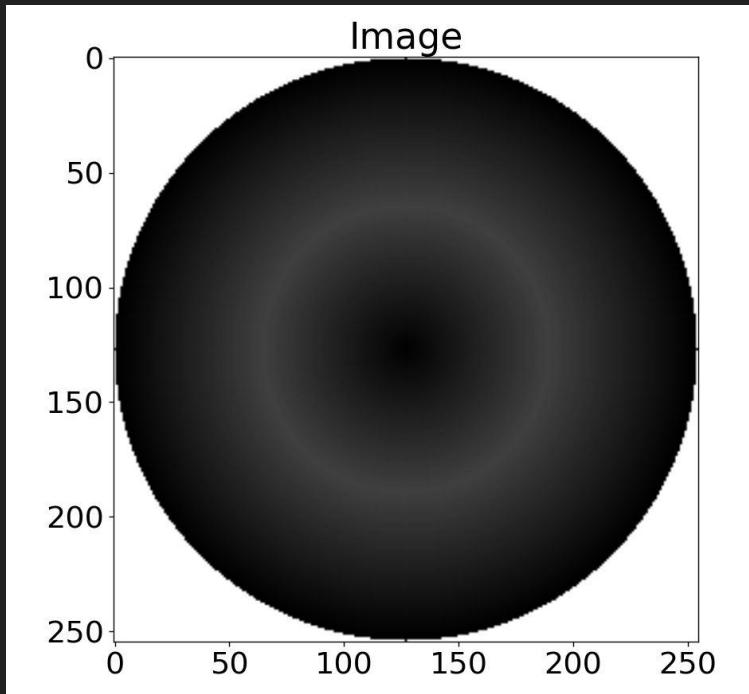
Some Images and Histograms



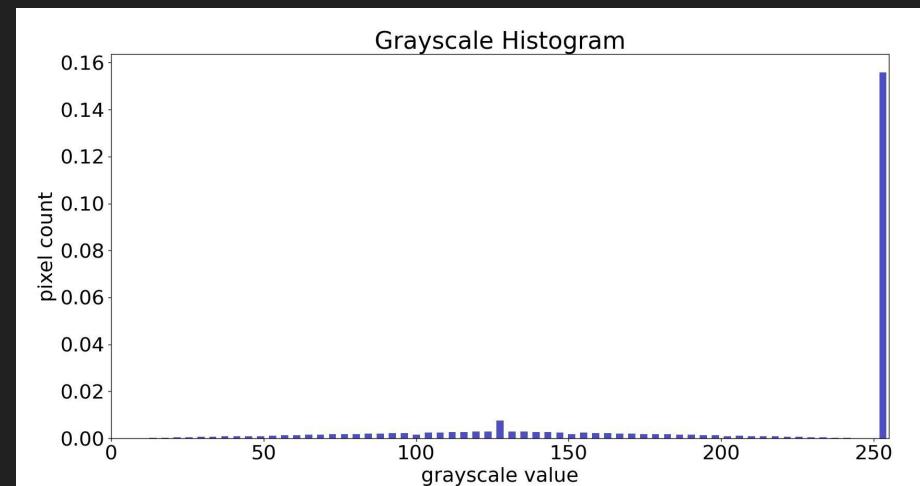
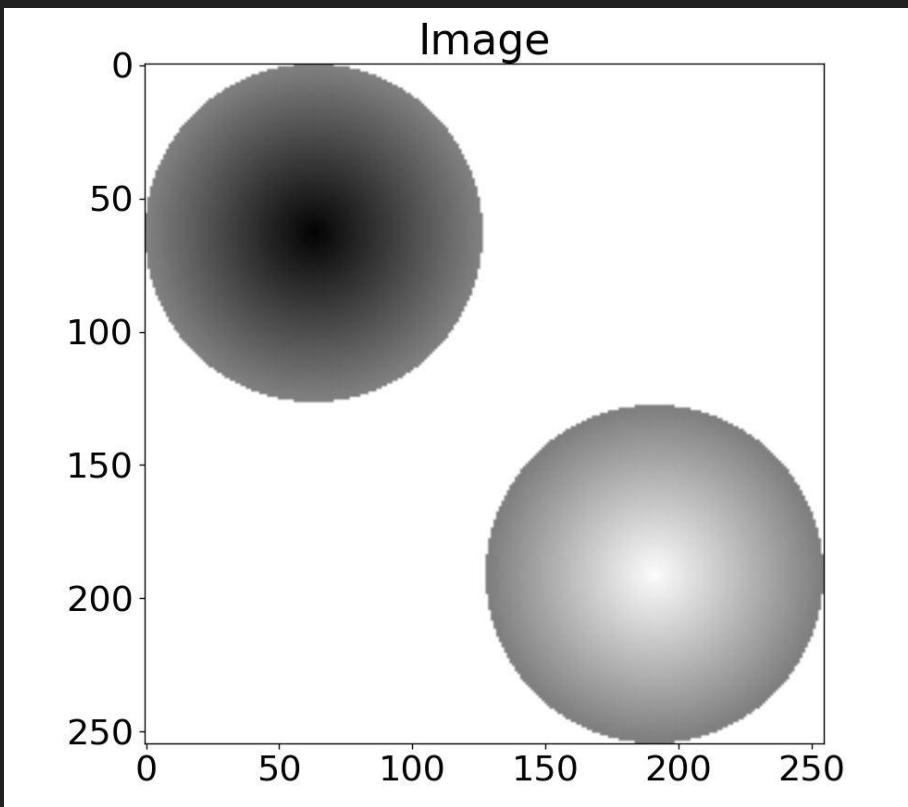
Some Images and Histograms



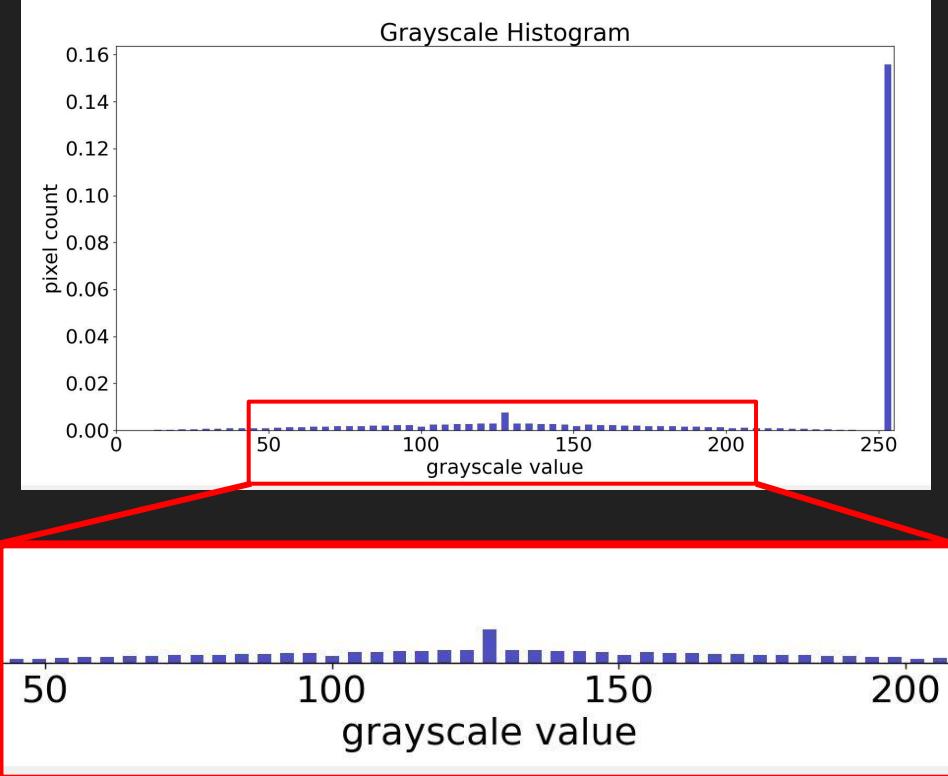
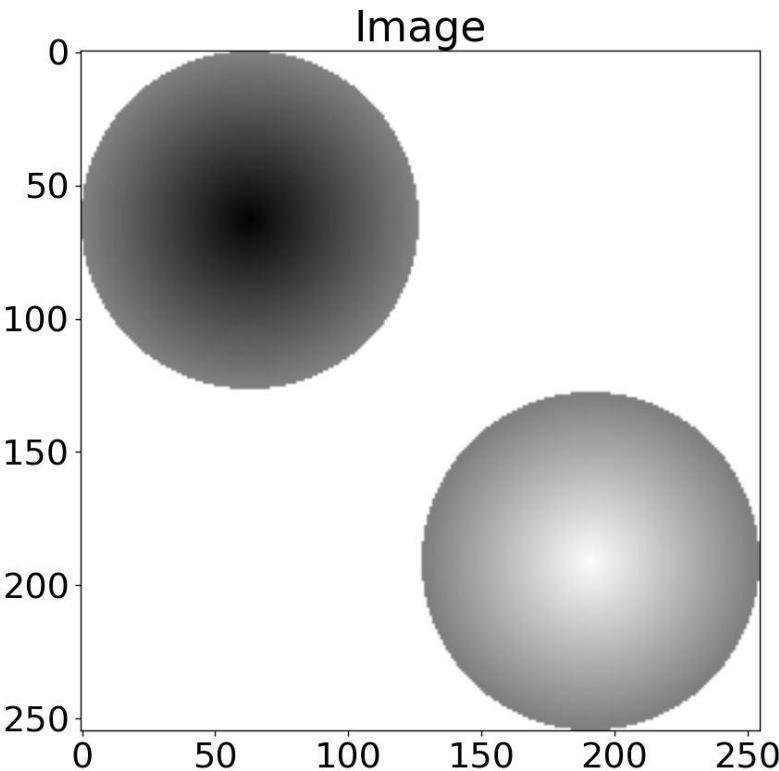
Some Images and Histograms



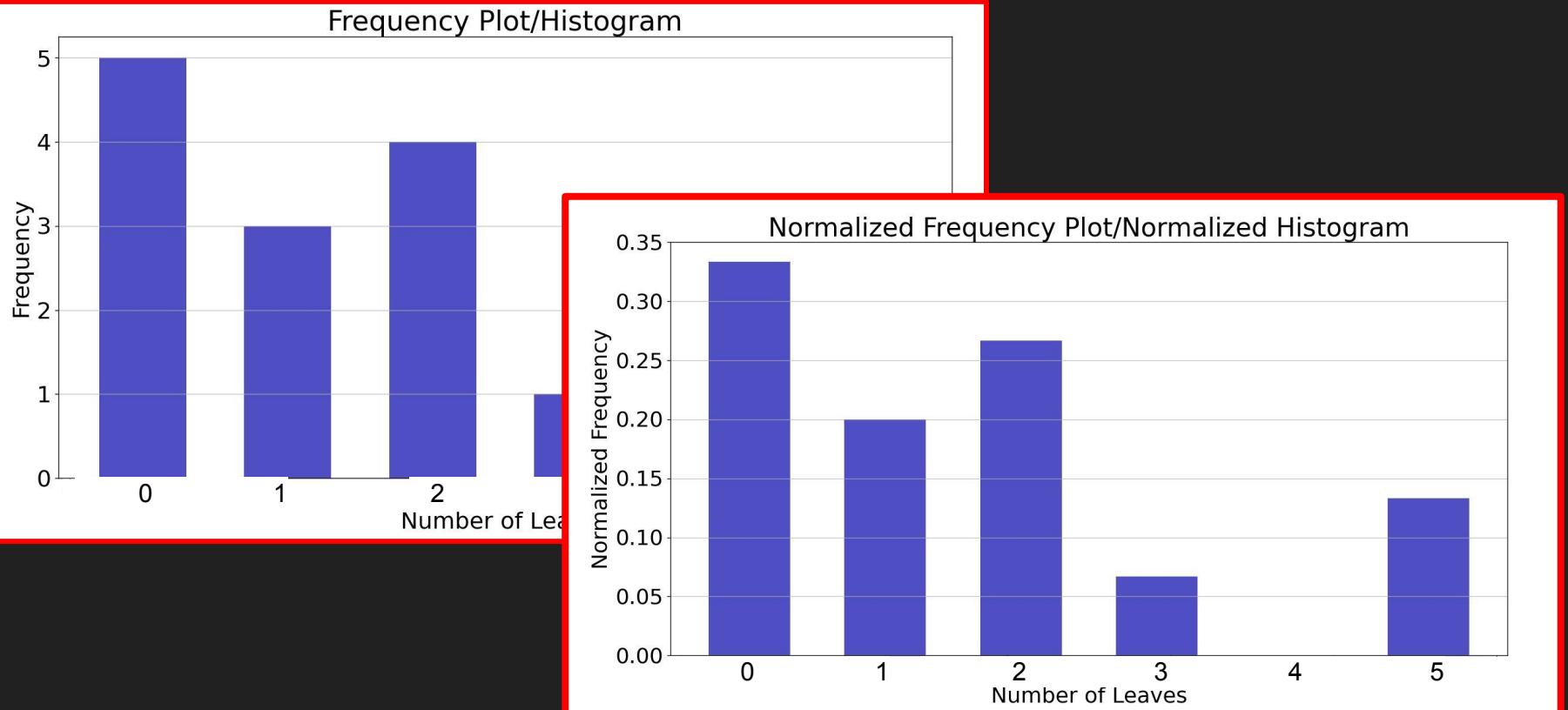
Some Images and Histograms



Some Images and Histograms



Describing Datasets: Why Normalization?



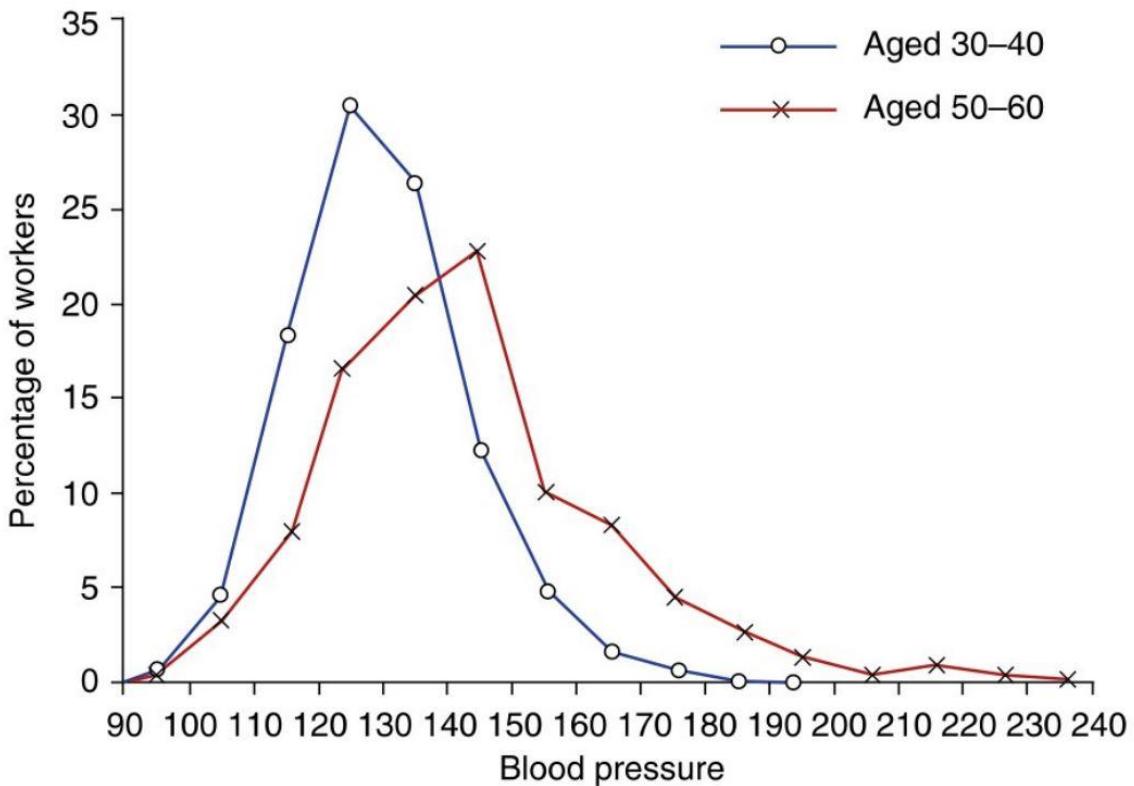
Describing Datasets: Why Normalization?

Blood pressure	Number of workers	
	Aged 30–40	Aged 50–60
Less than 90	3	1
90–100	17	2
100–110	118	23
110–120	460	57
120–130	768	122
130–140	675	149
140–150	312	167
150–160	120	73
160–170	45	62
170–180	18	35
180–190	3	20
190–200	1	9
200–210		3
210–220		5
220–230		2
230–240		1
Total	2540	731

Number of samples are unequal

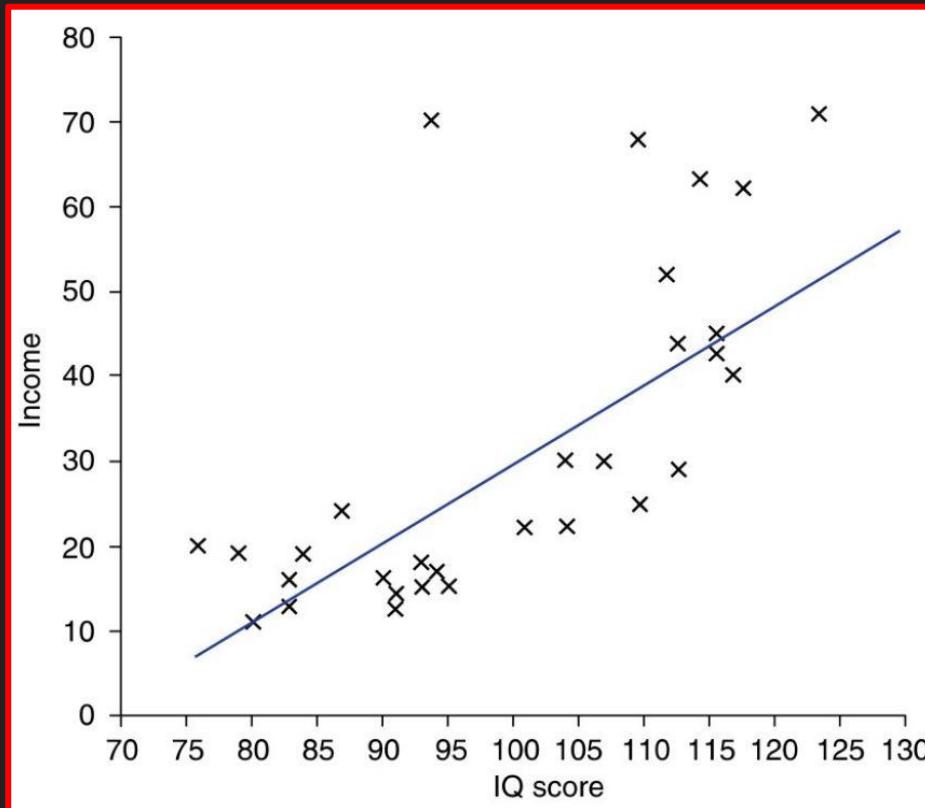
Blood pressure	Percentage of workers	
	Aged 30–40	Aged 50–60
Less than 90	0.12	0.14
90–100	0.67	0.27
100–110	4.65	3.15
110–120	18.11	7.80
120–130	30.24	16.69
130–140	26.57	20.38
140–150	12.28	22.84
150–160	4.72	9.99
160–170	1.77	8.48
170–180	0.71	4.79
180–190	0.12	2.74
190–200	0.04	1.23
200–210		0.41
210–220		0.68
220–230		0.27
230–240		0.14
Total	100.00	100.00

Describing Datasets: Frequency Polygons



Relative
Frequency Polygons

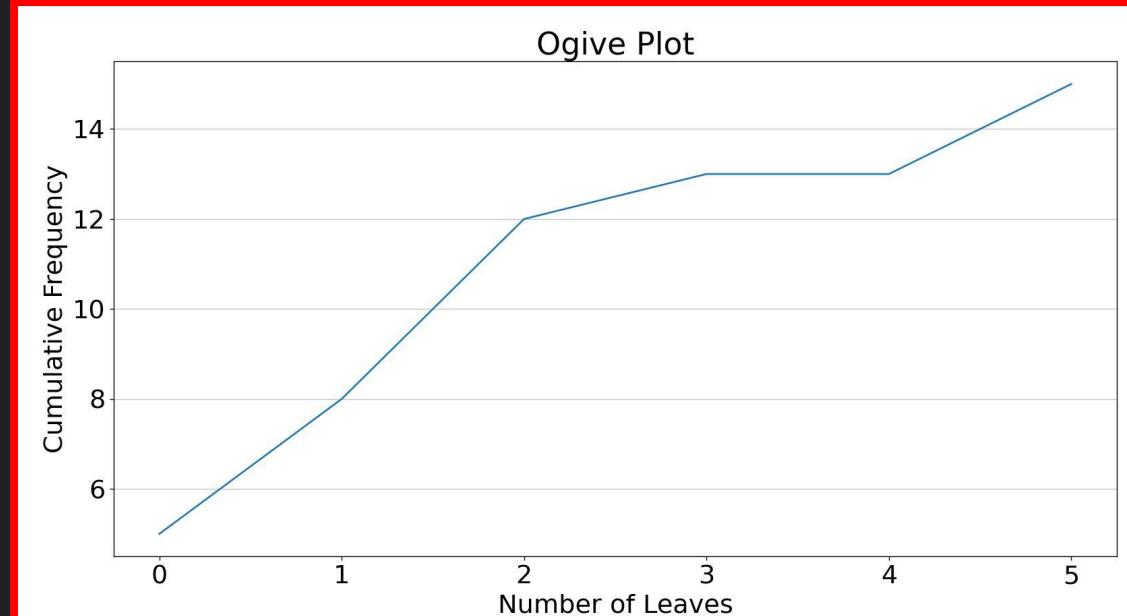
Describing Datasets: Scatter Plot



Ogive

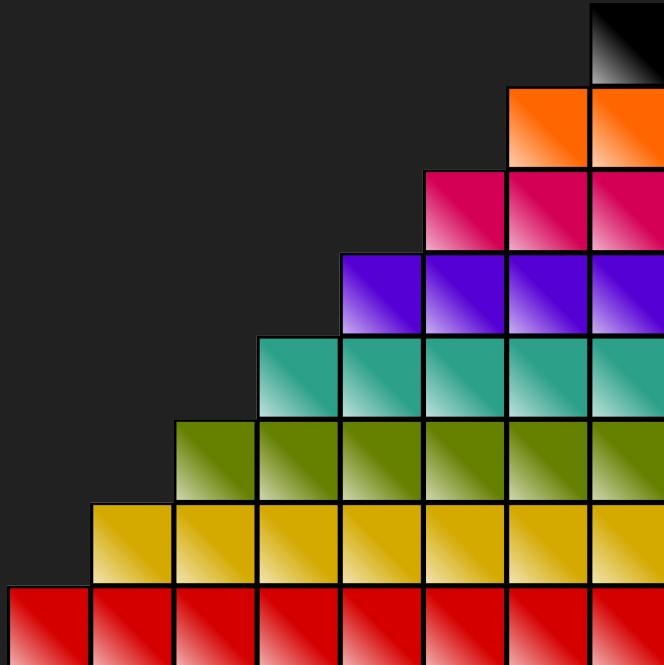
- Cumulative/relative frequency plot
- Proportion of number of sample points with value \leq given data value

Value	Frequency
0	5
1	3
2	4
3	1
4	0
5	2
Total	15



Mean and Median

Mean Height
4.5 units



Median Height
[4, 5] units

Mean

- Mean or Average is ratio of sum of numbers (x_1, x_2, \dots, x_N) and total numbers (N)
- Solution of $\min_x \sum_{i=1}^N (x - x_i)^2$
- Mean of [1, 2, 4] is 7/3
- Mean of [1, 2, 4, 6] is 13/4

Median

Percentile: value below which a percentage of data falls.

Example: You are the fourth tallest person in a group of 20

80% of people are shorter than you:



That means you are at the **80th percentile**.

If your height is 1.85m then "1.85m" is the 80th percentile height in that group.

Median

- Median is the value/values in the middle of the series of values
- To get median
 - First sort the values
 - Then find the value/values in the middle
- Solution of $\min_x \sum_{i=1}^N |x - x_i|$
- Mean of elements in [1, 2, 4] is 7/3, their Median is/are
 - 2
- Mean of elements in [1, 2, 4, 6] is 13/4, their Median is/are:
 - range [2, 4]

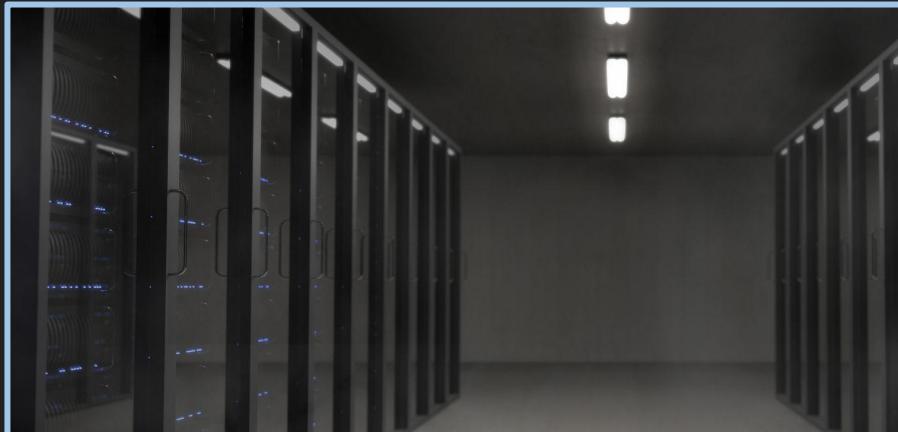
Median Vs Mean

Median has some advantages w.r.t. Mean.

- Who is better?
 - A team performing best consistently,
 - Or a team who performs bad all the times but outperform by large margin in just one game?
- Consider the array: [1, 1, 2, 2, 2]
- What will be its mean and median?
- Now, let's assume that there is a noise/mistake in the data:-
- [1, 1, 2, 2, 2, 100]
- 100 is added in the end by mistake
- Now, what will be mean and median?
- Median remains the same, i.e. it is less sensitive to the outliers (100 in our case).



Finding Temperature of Rooms



- N (=3) sensors in each room at different locations
- readings = [[12.4, 14.19, 23.19], [10.32, 20.14, 30.53]]
- Generate alarm if average temperature > 20
- Or should we use median?
- Should we treat fire at a corner as outlier!

Mode

- Most frequently occurring value
- 8, 10, 6, 4, 10, 12, 14, 10
- Sample mode is 10
- Need not be unique
- Must be among one of the values in the data

Deviations

Deviations from the mean: $x_i - \bar{x}$

HW: Show that:

$$\sum_{i=1}^n (x_i - \bar{x}) = 0$$

Standard Deviation (σ) and Variance

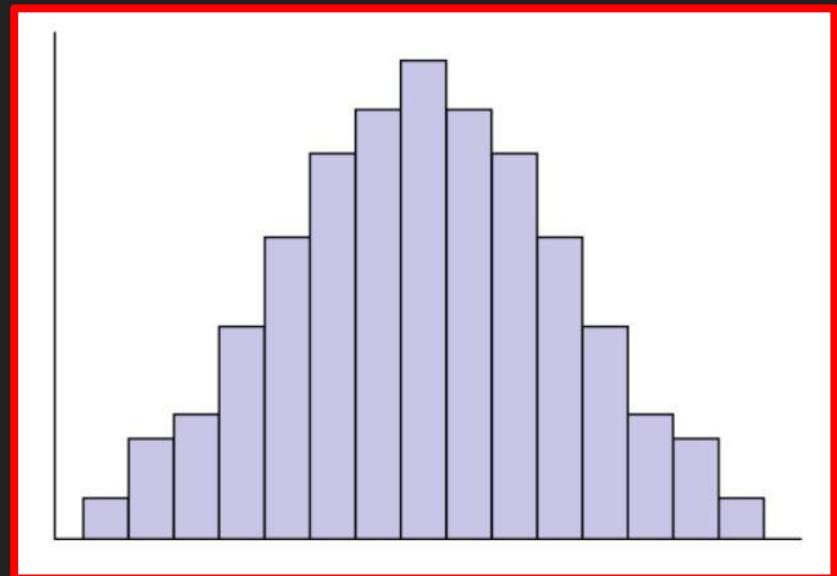
- A: 1,2,5,6,6
- B: -40,0,5,20,35
- Same mean, but B has more spread
- Sample standard deviation (σ)
- σ = second order moment/deviation from mean
- σ = positive square root of sample variance
- Variance = σ^2

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

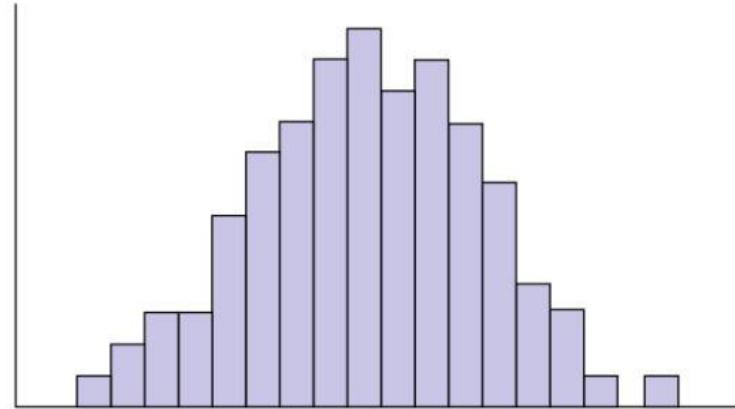
Normal Dataset

A dataset is normal if its histogram:

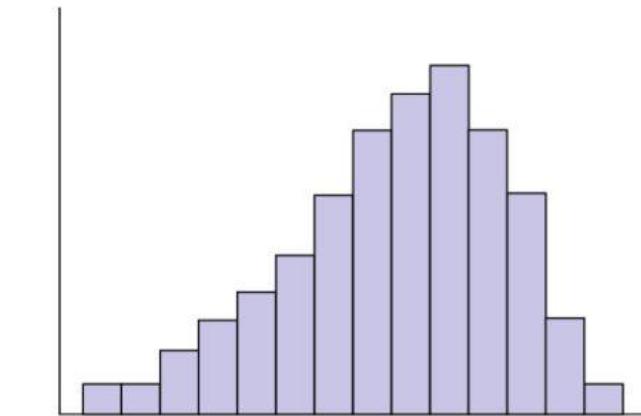
- Is highest at the middle interval
- Is bell-shaped
- Is symmetric about its middle interval



Normal Dataset



Approximately normal



Skewed

Normal Dataset: Empirical Rule

1. Approximately 68% of the data lie within

$$\bar{x} \pm s$$

mean = \bar{x}

standard deviation
= s

Normal Dataset: Empirical Rule

1. Approximately 68% of the data lie within

$$\bar{x} \pm s$$

2. Approximately 95% of the data lie within

$$\text{mean} = \bar{x}$$

$$\begin{aligned}\text{standard deviation} \\ = s\end{aligned}$$

$$\bar{x} \pm 2s$$

Normal Dataset: Empirical Rule

1. Approximately 68% of the data lie within

$$\bar{x} \pm s$$

mean = \bar{x}

2. Approximately 95% of the data lie within

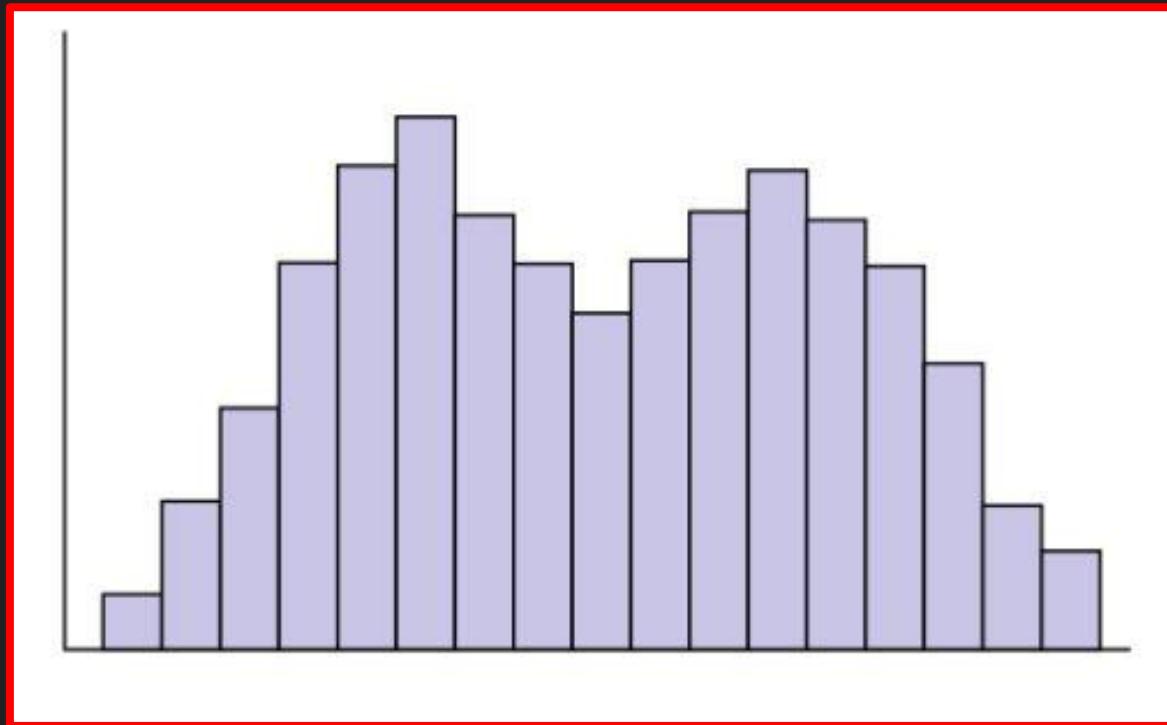
$$\bar{x} \pm 2s$$

standard deviation
= s

3. Approximately 99.7% of the data lie within

$$\bar{x} \pm 3s$$

Bimodal Data



Sample Correlation Coefficient (r_{xy})

$$X = [x_1 - \bar{x}, x_2 - \bar{x}, \dots, x_n - \bar{x}]$$

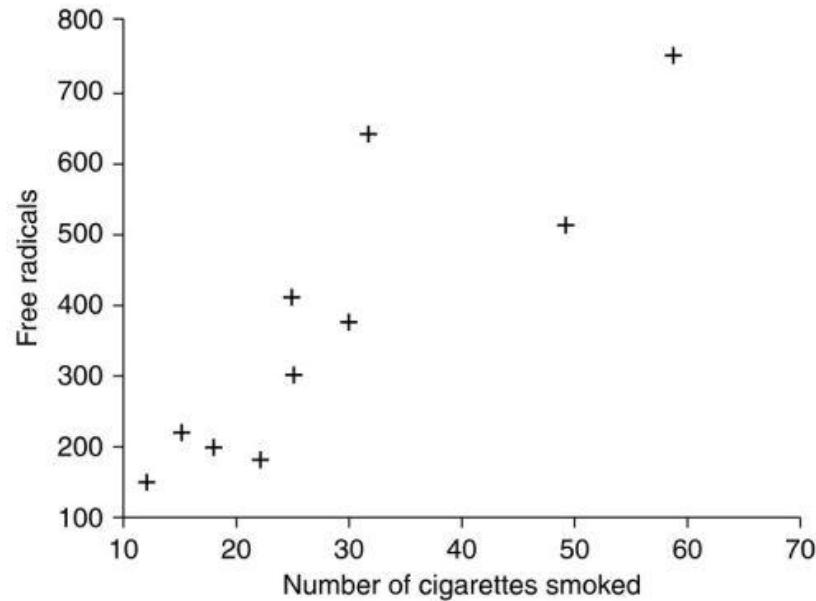
$$Y = [y_1 - \bar{y}, y_2 - \bar{y}, \dots, y_n - \bar{y}]$$

What will be ratio of $X \cdot Y$ and $|X||Y|$?

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{(n - 1)s_x s_y}$$

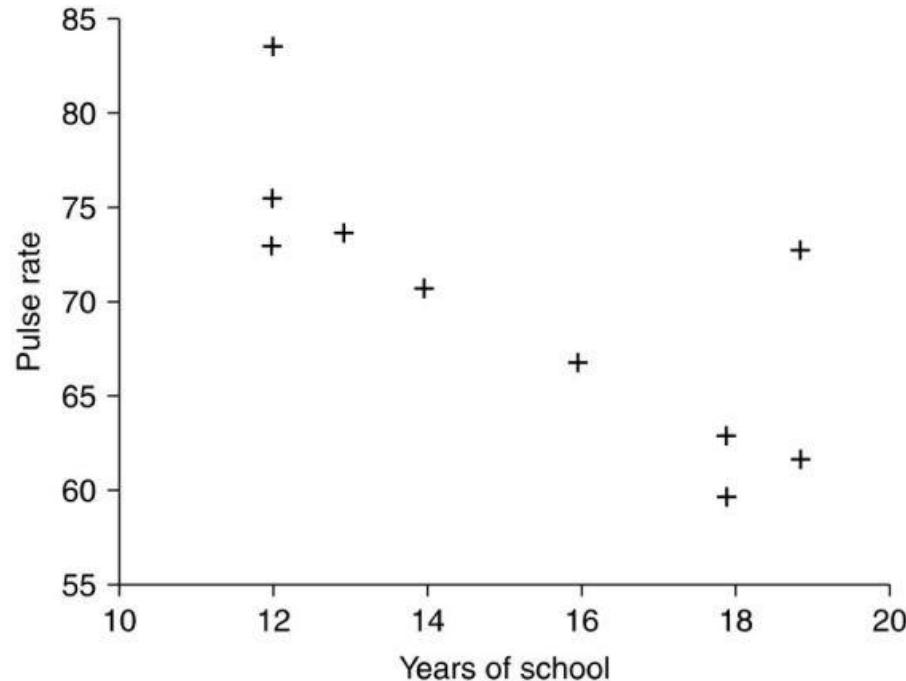
Sample Correlation Coefficient (r_{xy})

Person	Number of cigarettes smoked	Free radicals
1	18	202
2	32	644
3	25	411
4	60	755
5	12	144
6	25	302
7	50	512
8	15	223
9	22	183
10	30	375



Sample Correlation Coefficient (r_{xy})

	Person									
	1	2	3	4	5	6	7	8	9	10
Years of school	12	16	13	18	19	12	18	19	12	14
Pulse rate	73	67	74	63	73	84	60	62	76	71



Sample Correlation Coefficient (r_{xy})

- Large x_i 's correlated to large y_i 's, and small x_i 's correlated to small y_i 's
- Strong negative correlation between number of years in school and resting pulse rate
- Does this that imply more years in school reduces the pulse rate?
- Correlation does not necessarily imply causation
- Sleeping with one's shoes on is strongly correlated with waking up with a headache. Therefore, sleeping with one's shoes on causes headache. Missing factor: going to bed drunk.
- As ice cream sales increase, the rate of drowning deaths increases sharply. Therefore, ice cream consumption causes drowning. Missing factor: summer weather.

Lambda Notation

```
sorted_myDict = sorted(myDict.items(), key=lambda x:x[1], reverse=True)
```

Some Questions

- If each point in $X = [x_1, x_2, \dots, x_n]$ with mean μ_x and variance v_x
- is changed to $[ax_1+b, ax_2+b, \dots, a x_n + b]$
- What will be the new mean and variance?

Chebyshev's Inequality

- If the average score for a course with 300 students is 40 and standard deviation is 5, how many students will get marks between 30 and 50?
- We cannot tell exactly.
- But, we can tell in terms of inequality.
- If $|S_k|$ number of points follow $|x_i - \mu_x| > k \sigma_x$
- Then, $|S_k| / N < k^{-2}$
- Similarly,
- If $|S_k|$ number of points follow $|x_i - \mu_x| \geq k \sigma_x$
- Then, $|S_k| / N \leq k^{-2}$

Chebyshev's Inequality

Therefore,

- If,
- $S_k = \{x_i : |x_i - \mu_x| < k \sigma_x\}$
- Then,
- $|S_k|/N > (1 - k^{-2})$
- \leq in If will change $>$ to \geq in then

Chebyshev's Inequality

Proof, $S_k = \{x_i : |x_i - \mu_x| > k \sigma_x\}$

$$(\sigma_x)^2 = (n-1)^{-1} \sum (x_i - \mu_x)^2$$

$$= (n-1)^{-1} (\sum (x_i^S - \mu_x)^2 + \sum (x_i^C - \mu_x)^2), \text{ where } x_i^S \in S_k, \text{ and } x_i^C \notin S_k$$

$$> (n-1)^{-1} (\sum (k\sigma_x)^2 + \sum (x_i^C - \mu_x)^2)$$

$$> (n-1)^{-1} (|S_k|(k\sigma_x)^2 + \sum (x_i^C - \mu_x)^2)$$

$$> (n-1)^{-1} |S_k| (k\sigma_x)^2$$

$$\text{So, } (n-1) (\sigma_x)^2 > |S_k| (k\sigma_x)^2$$

Chebyshev's Inequality

Therefore,

- If,
- $S_k = \{x_i : x_i - \mu_x \geq k \sigma_x\}$
- Then,
- $|S_k|/N \leq 1/(1 + k^2)$
- Proof: Homework
- Hint Start with $\sum (x_i - \mu_x + b)^2$, where b is constant and $b > 0$

Basics of Probability

Tossing a coin, what is practical significance of value 0.5?





Basics of Probability

Tossing two coins.

- Experiments: any process that produces an outcome
- Outcomes: An observation
- Sample space: set of all outcomes. $S = \{(HH), (HT), (TH), (TT)\}$
- Random experiment: outcome is not predictable
- Event: A set of outcomes
 - $A = \{(HH), (HT)\}$: A is the event that the first coin lands on heads
 - $B = \{(HT), (TT)\}$: B is the event that the second coin lands on tails
 - $A \cap B = \{(HT)\}$: event that the first coin landed heads and second landed tails

Basics of Probability

Tossing a coin, what is practical significance of value 0.5?

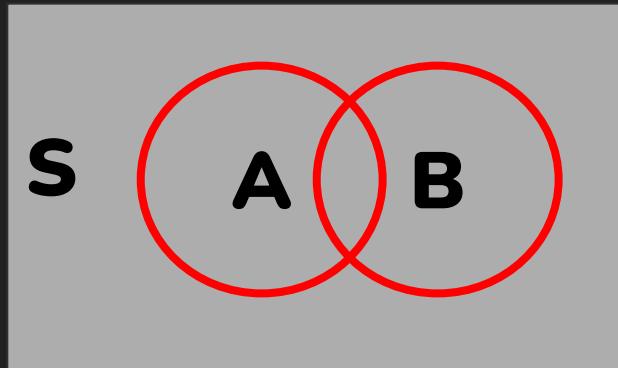


Basics of Probability



E.g., Tossing two coins.

- Sample space: set of all outcomes. $S = \{(HH), (HT), (TH), (TT)\}$
- Event: A set of outcomes
 - $A = \{(HH), (HT)\}$: first coin lands on heads
 - $B = \{(HT), (TT)\}$: second coin lands on tails
 - $A \cap B = \{(HT)\}$
 - $A \cup B = S - \{TH\}$



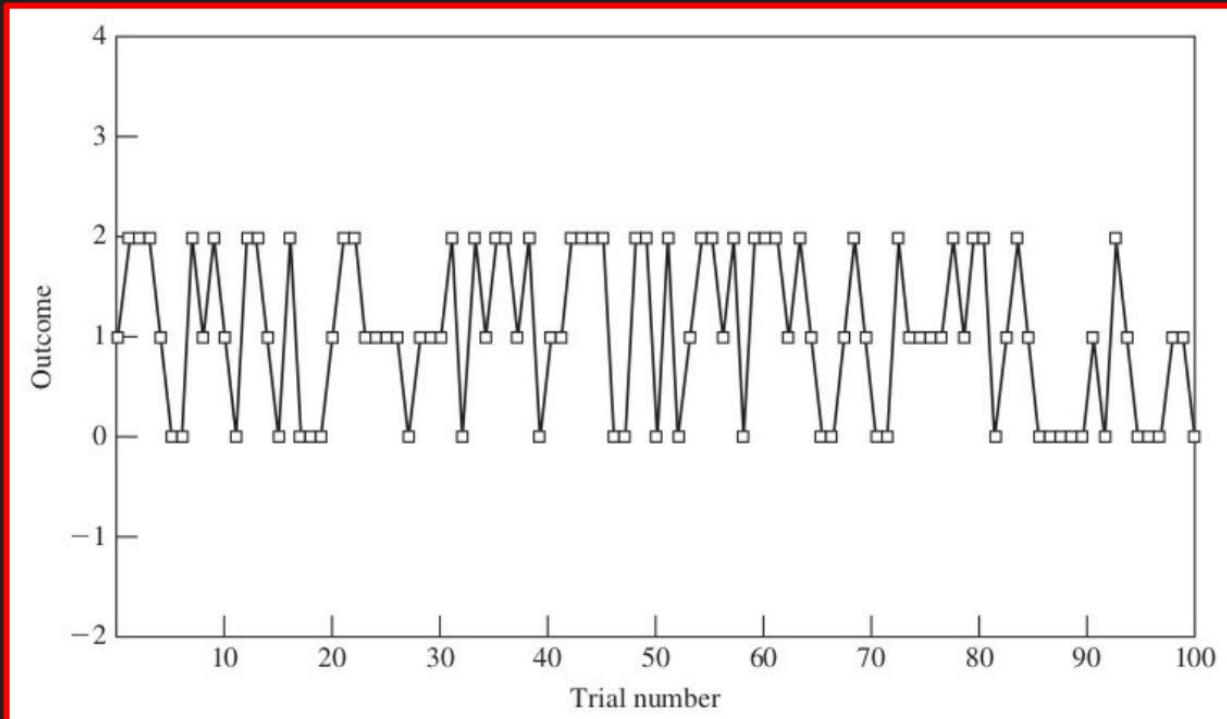
Basics of Probability

- Urn with 3 Balls
- Urn shaken, ball picked,
number noted.
- What is the outcome of this
experiment?
- Number from set $S = \{1, 2, 3\}$
- Probability models depend on statistical regularity.



Basics of Probability

- Urn with 3 Balls
- Urn shaken, ball picked, number noted.
- Statistical Regularity.



- Averages obtained in long sequences of repetitions yield the same value.

Basics of Probability



- Repeat Urn experiment n times under identical conditions.
- Number of times ball k appears:
 $N_k(n)$, $k \in \{1, 2, 3\}$
- Relative frequency of outcome k:

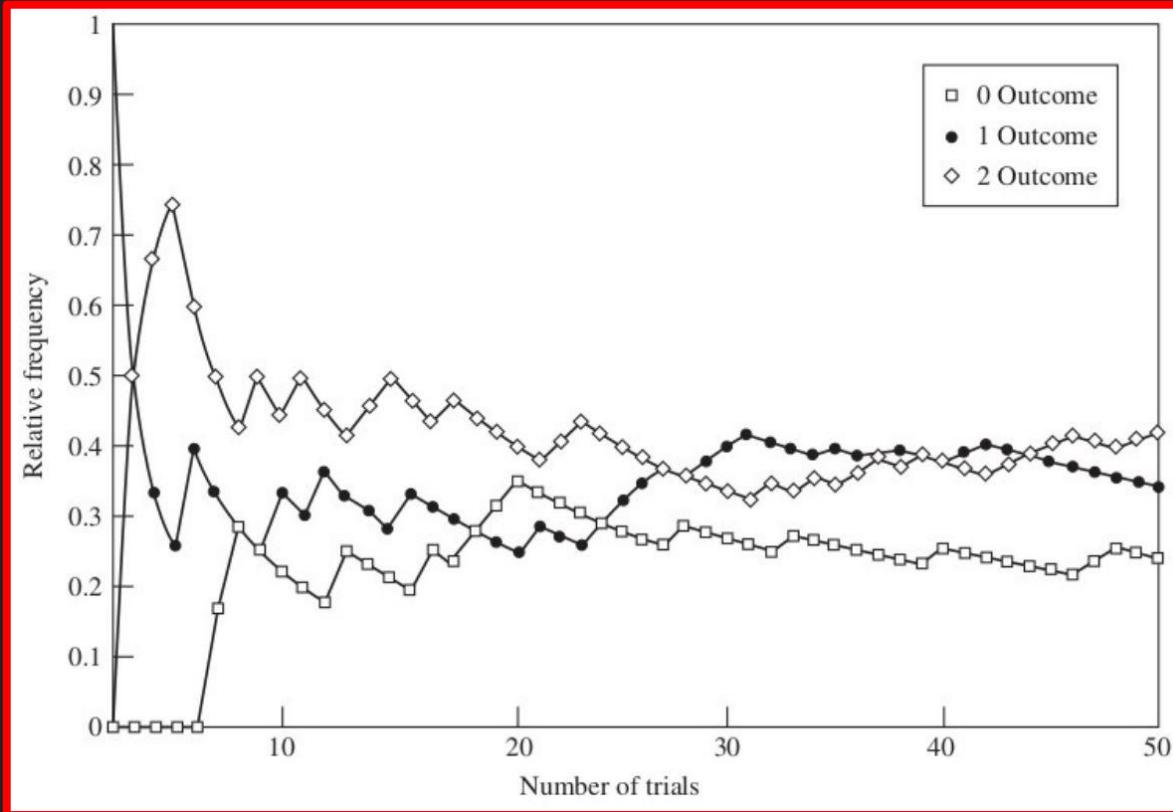
$$f_k(n) = \frac{N_k(n)}{n}$$

$$\lim_{n \rightarrow \infty} f_k(n) = p_k$$

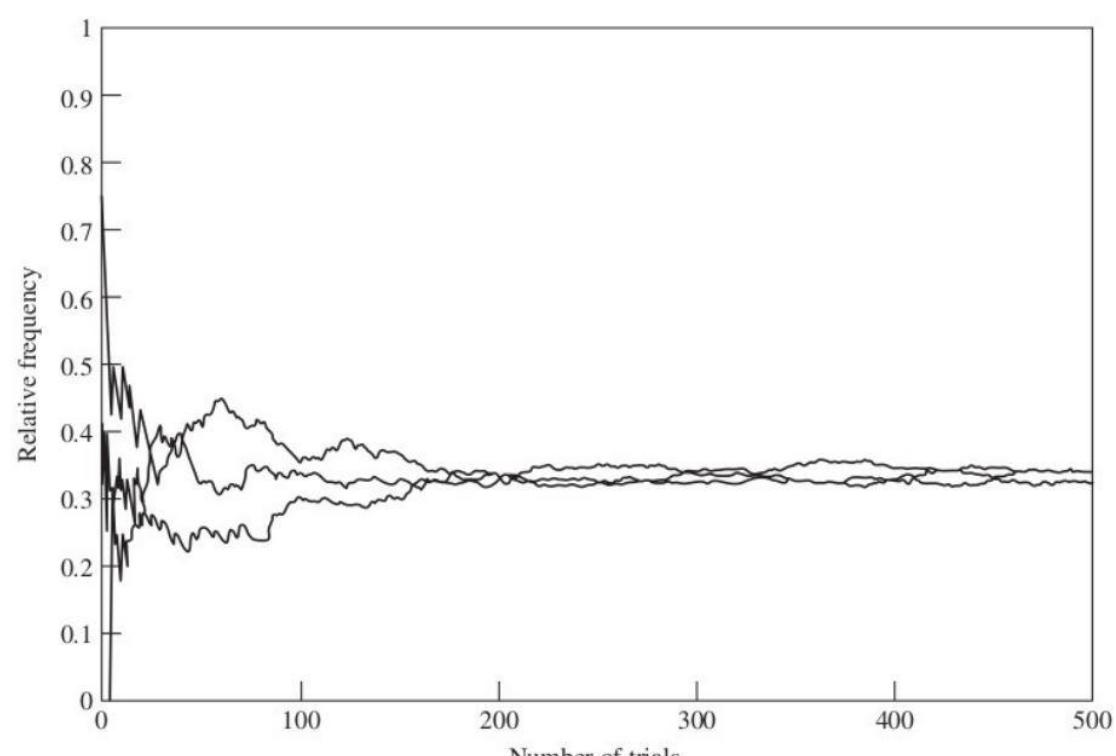
$$p_k = 1/3$$

- Averages obtained in long sequences of repetitions yield the same value.

Basics of Probability



Basics of Probability



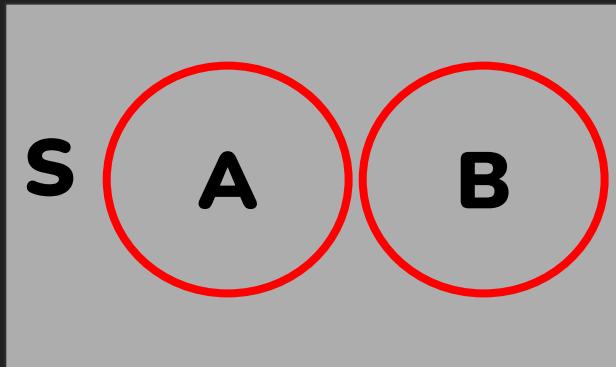
Basics of Probability

- Consider n trials of a random experiment with K possible outcomes.
- $S = \{1, 2, \dots, K\}$
- $0 \leq N_k(n) \leq n$, for $k = 1, 2, \dots, K$
- So relative frequency $f_k(n)$: $0 \leq f_k(n) \leq 1$
- Also, $\sum_k N_k(n) = n$
- So, $\sum_k f_k(n) = 1$

Basics of Probability



- Let C be the event “A or B occurs”
- where A and B are events which cannot occur simultaneously (i,e, A and B are disjoint)
- Then,
- $f_C(n) = f_A(n) + f_B(n)$



Axiomatic Approach to Probability

Probability was defined by its long-term relative frequency

- Several problems with this definition:
 - Limit may not be defined
 - We cannot perform an experiment infinite times, therefore p_k is an approximation
 - What about experiments that can't be repeated?
- Thus, we need another definition, independent of the application
- Also, we must be able to interpret probability intuitively as relative frequency

Axioms of Probability

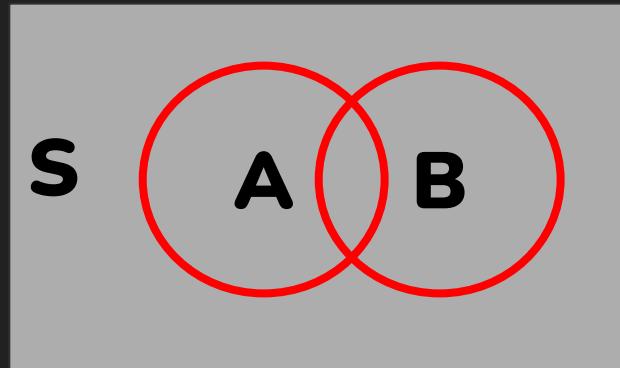
Let us assume:

- A random experiment has been defined
- The set of all possible outcomes S has been defined
- Subsets of S called events have been defined
- Each event A has been assigned a number $P[A]$, such that:
 - Axiom 1: $P[A] \geq 0$
 - Axiom 2: $P[S] = 1$
 - Axiom 3: $P[A_1 \cup A_2 \cup \dots \cup A_n] = P[A_1] + P[A_2] + \dots + P[A_n]$, if A_1, A_2, \dots, A_n are disjoint events.



Basics of Probability

- Demorgan's law:
- $A^c \cup B^c = (A \cap B)^c$
- Boole's Inequality:
- $P(A \cup B) \leq P(A) + P(B)$
- Bonferroni inequality:
- $P(A \cap B) \geq P(A) + P(B) - 1$
- Proof:
- $P(A^c \cup B^c) \leq P(A^c) + P(B^c)$



Axioms of Probability

Let us assume:

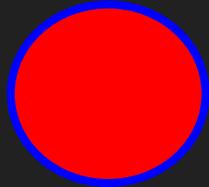
- A random experiment has been defined
- The set of all possible outcomes S has been defined
- Subsets of S called events have been defined
- Each event A has been assigned a number $P[A]$, such that:
 - Axiom 1: $P[A] \geq 0$
 - Axiom 2: $P[S] = 1$
 - Axiom 3: $P[A_1 \cup A_2 \cup \dots \cup A_n] = P[A_1] + P[A_2] + \dots + P[A_n]$, if A_1, A_2, \dots, A_n are disjoint events.
- We are not worried about what the probabilities mean, or how they are obtained.
- This depends on the user or application.

Application Example



- Telephone conversation: is the speaker talking or is silent?
- Typical speaker only active 1/3rd of the time (someone has to give this to you; eg. a domain expert.)
- Random experiment:
- Urn with 2 white balls (silence) and 1 black ball (speech)
- Question: What is the probability that more than 24 out of 48 independent speakers are active at the same time?
- Model: what is the probability that 24 black balls are selected in 48 independent repetitions of the urn experiment?

Random Experiments and Event Class



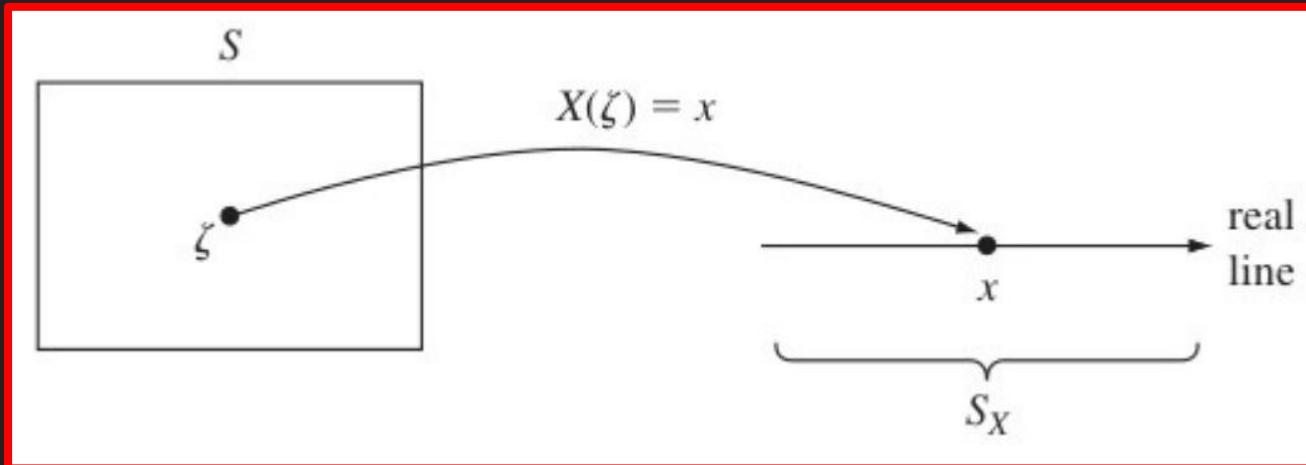
- Sample Space S : All possible outcomes of a random experiment.
- Define the event class \mathcal{F} of all events of interest. Remember: event is set of outcomes.
- Events in \mathcal{F} are assigned probabilities.
- \mathcal{F} is closed under complements, countable unions and intersections.
- When S is finite or countable, \mathcal{F} is powerset (set of all subsets) of S .
- When S is uncountable (e.g., \mathbb{R}), \mathcal{F} cannot be set of all subsets of S .
- We can still model events of practical interest by countable unions and intersections of intervals of \mathbb{R} .

Random Variables

- In many random experiments, we may not be interested in the observed values, but in some numerical quantity determined by the observed values.
- Examples:
- Sum of values of two dice throws
- Number of tails appearing in n consecutive coin tosses
- Random variables are any such quantities determined by the results of random experiments.
- They may be observations themselves.

Random Variables

A random variable is a function that assigns a real number to each outcome in the sample space.



What are domain and range of X ?

Random Variables Notation

Random variable:

- Usually denoted as upper case alphabet, e.g., X .
- Individual values it can acquire are denoted by lowercase, e.g., x .

Random Variables Example

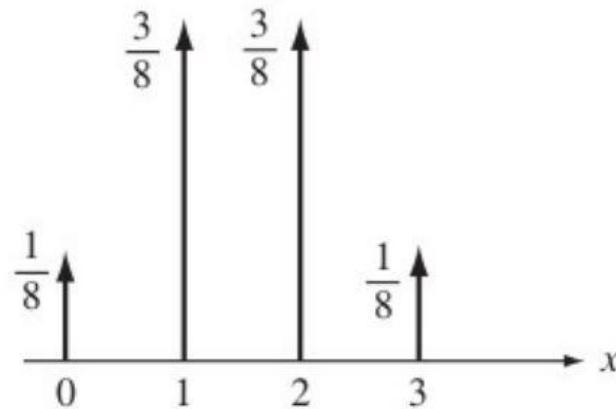
Value of X (denoted as x) X = Sum of 2 dice throws.	Probability Mass Function (pmf) of X $P(X = x)$
2	1/36
3	2/36
4	3/36
5	
.	
.	
.	
12	1/36

- If S is the sample space,
- Then,
- $P[S] = P(\text{Union of all events of form } X=x) = 1$
- Complete the table and verify.

Discrete Random Variables

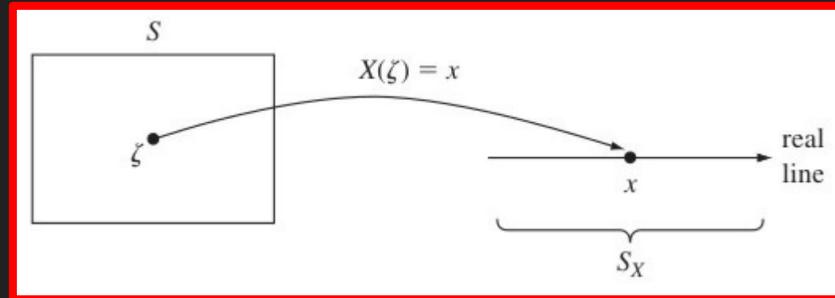
- Random variables whose values can be written as finite or infinite sequence.
- Probability Mass Function $P(X=x)$ is the probability that random variable X can take value x.
- It can also be written as $p_x(x)$ or sometimes $p(x)$.
- Examples?
- Cumulative Distribution Function (cdf) is defined as $F_x(x) = P(X \leq x)$.

Discrete Random Variables



What random variable does this PMF represent?

Expected Values



- The expected value of X is

$$E[X] = \mu = \sum_{x \in S_X} xP(x)$$

Contrast this with
the sample mean
and sample
variance

- The variance is

$$\sigma^2 = E[(x - \mu)^2] = \sum_{x \in S_X} (x - \mu)^2 P(x)$$

Expected Values

Find Expected value of
following pmf:

- The expected value of X is

$$E[X] = \mu = \sum_{x \in S_X} xP(x)$$

Contrast this with
the sample mean
and sample
variance

- The variance is

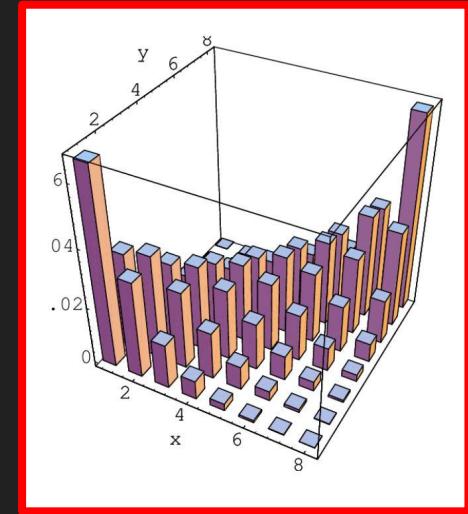
$$\sigma^2 = E[(x - \mu)^2] = \sum_{x \in S_X} (x - \mu)^2 P(x)$$

$$P(X=x) = k / x^2 \text{ for } x \geq 1, x \in \mathbb{Z}^+$$

What will be value of constant k ?

Pairs of Discrete Random Variables

- Two random variables X and Y taking values from S_X and S_Y
- $S_X = \{x_1, x_2, \dots, x_n\}$
- $S_Y = \{y_1, y_2, \dots, y_n\}$
- (x, y) is point in $S_X \times S_Y$
- Joint probability $p_{ij} = P(X = x_i, Y = y_j)$
- Marginal distributions of X and Y are:



$$P_X(x) = \sum_{y \in S_Y} P(x, y)$$

$$P_Y(y) = \sum_{x \in S_X} P(x, y)$$

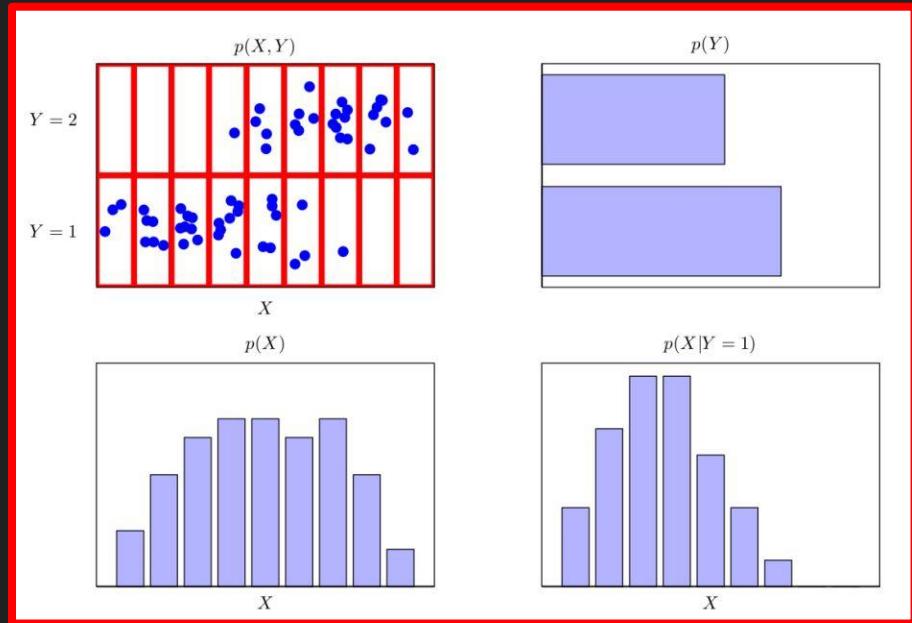
Joint PMFs

Given two r.v. X and Y , their joint probability mass function is defined as:

- $p_{X,Y}(x_i, y_j) = P(X = x_i, Y = y_j)$
- Marginal pmf is obtained by:
 - $P(\{X = x_i\})$
 - $= P(\bigcup_j \{X = x_i, Y = y_j\})$
 - $= \sum_j P(\{X = x_i, Y = y_j\})$
 - $= \sum_j p_{X,Y}(x_i, y_j)$

Joint Distributions and Conditional Distributions

- 60 samples drawn from joint distribution (top left figure)
- How many values X can take?
- How many values Y can take?
- Histogram Estimate of Marginal distribution of X.
- Histogram Estimate of Marginal distribution of Y.
- Conditional distribution $P[X|Y=1]$
- $p_{X|Y}(x|y) = p_{XY}(x,y) / p_Y(y)$



Independence

- Two random variables X and Y are independent iff:

$$p_{XY}(x,y) = p_X(x) p_Y(y) \text{ for all } x \text{ and } y$$

- If $X = x$ is given, then if fraction of situations in which $Y = y$ is still $P_Y(y)$.
- Then knowing X does not give any additional knowledge about Y.
- Then, X and Y are independent.
- E.g.
- Mark drives at 60 km/h to his office, I have 3 chapatis for dinner.
- E.g. (Not Independent):
- You go to a shopping mall with multiplex, You eat popcorn.

Joint Distribution and Independence

Value of X (denoted as x) X = Number on 1 st Dice	Probability Mass Function (pmf) of X $P(X = x)$	Value of Y (denoted as y) Y = Number on 2 nd Dice	Probability Mass Function (pmf) of Y $P(Y = y)$	Value of (X,Y) (denoted as x,y)	Joint Distribution on (X,Y), $P(X=x,$ $Y=y)$
1	1/6	1	1/6	1, 1	1/36
2	1/6	2	1/6	2, 1	1/36
3	1/6	3	1/6	3, 1	1/36
4	1/6	4	1/6	.	.
4	1/6	4	1/6	.	.
6	1/6	6	1/6	6, 6	1/36

Is $p_{XY}(x,y) = p_X(x) p_Y(y)$ for all x and y ?

Two rules of probability

Sum rule

$$P(x) = \sum_Y P(x, y)$$

Two rules of probability

Sum rule

$$P(x) = \sum_Y P(x, y)$$

Product rule

$$P(x, y) = P(y|x)P(x)$$

Bayes Rule

Using the fact that $P(x,y) = P(y,x)$, and the sum and product rules, we get **Baye's rule**:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Using the sum rule and product rule, derive Bayes rule.

False Positive Paradox

A drug test proposed by company:

- Tests +ve 99 % of the time on drug consumers, and
- Tests -ve 99 % of the time on non-consumers
- Consumers are 0.5 % of people
- If a person tests +ve, what is the probability that he/she is actually a consumer?
- Let C be the event that person is a drug consumer
- Let $+$ be the event that person tests +ve
- $P(C|+) = P(+|C) P(C) / P(+) = 0.99 * 0.005 / P(+)$
- $P(+) = P(+|C) P(C) + P(+|C^c) P(C^c) = 0.99 * 0.005 + (1-0.99)*0.995$
- $P(C|+) = ?$
- Is the person testing positive more likely a consumer or a non-consumer?
- More examples on https://en.wikipedia.org/wiki/Base_rate_fallacy

Binomial Random Variable

$$S_X = \{0, 1, 2, \dots, n\}$$

$$p_k = \binom{n}{k} p^k (1-p)^{n-k} \quad k = 0, 1, \dots, n$$

$$E[X] = np \quad \text{var}[X] = np(1-p)$$

X is the number of successes in n iid trials, with the probability of success in each trial being p .

Sampling data from Binomial Distribution

```
from numpy.random import default_rng
import numpy as np
import matplotlib.pyplot as plt

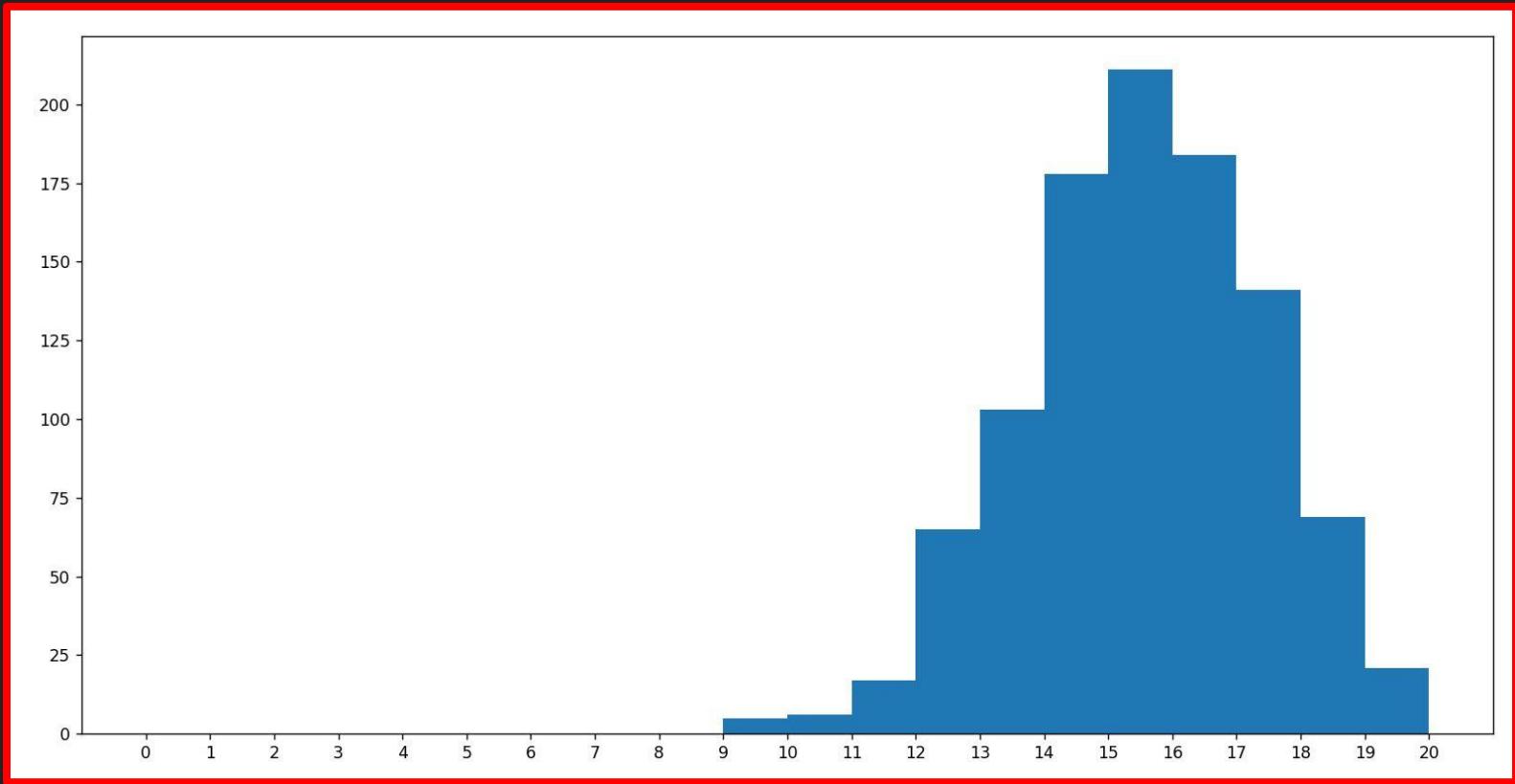
# binomial distribution
n, p = 20, 0.75 # no. of trials, and prob. of each trial

# repeat experiment of 20 trials, 1000 times.
rng = default_rng()
s = rng.binomial(n, p, 1000) #[15 17 18 14 ... 17], len(s) = 1000

myBins = np.arange(0, 21, 1)#[0 1 2 ... 20]

plt.hist(s, bins = myBins)
plt.xticks(myBins)
plt.show()
```

Sampling data from Binomial Distribution



Poisson Random Variable

$$S_X = 0, 1, 2, \dots$$

$$p_k = \frac{\alpha^k}{k!} e^{-\alpha} \quad k = 0, 1, \dots \quad \text{and} \quad \alpha > 0$$

$$E[X] = \alpha \quad \text{var}[X] = \alpha$$

X is the number of events that occur in one time unit when time between events is exponentially distributed with mean $1/\alpha$.

Sampling data from Poisson Distribution

```
from numpy.random import default_rng
import numpy as np
import matplotlib.pyplot as plt

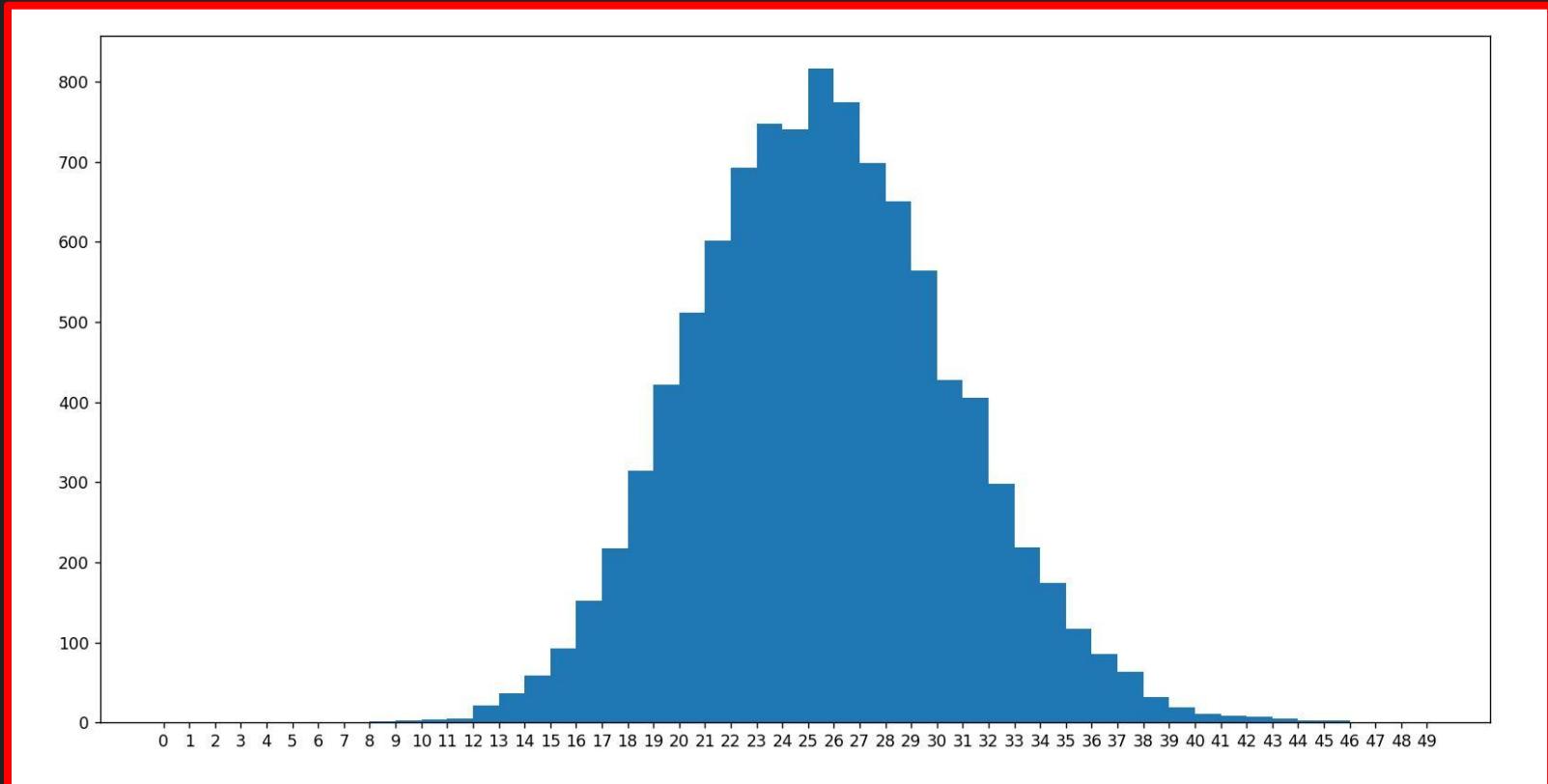
# poisson distribution
alpha = 25 # mean, max value of no of events per unit time

# repeat experiment 10000 times.
rng = default_rng()
s = rng.poisson(alpha, 10000) #len(s) = 10000

myBins = np.arange(0, 50, 1) # [0 1 2 ... 49]

plt.hist(s, bins = myBins)
plt.xticks(myBins)
plt.show()
```

Sampling data from Poisson Distribution



Uniform (Discrete) Random Variable

$$S_X = 0, 1, 2, \dots, L-1$$

$$p_k = \frac{1}{L} \quad k = 0, 1, \dots, L-1$$

Sampling data from Uniform Distribution

```
from numpy.random import default_rng
import numpy as np
import matplotlib.pyplot as plt

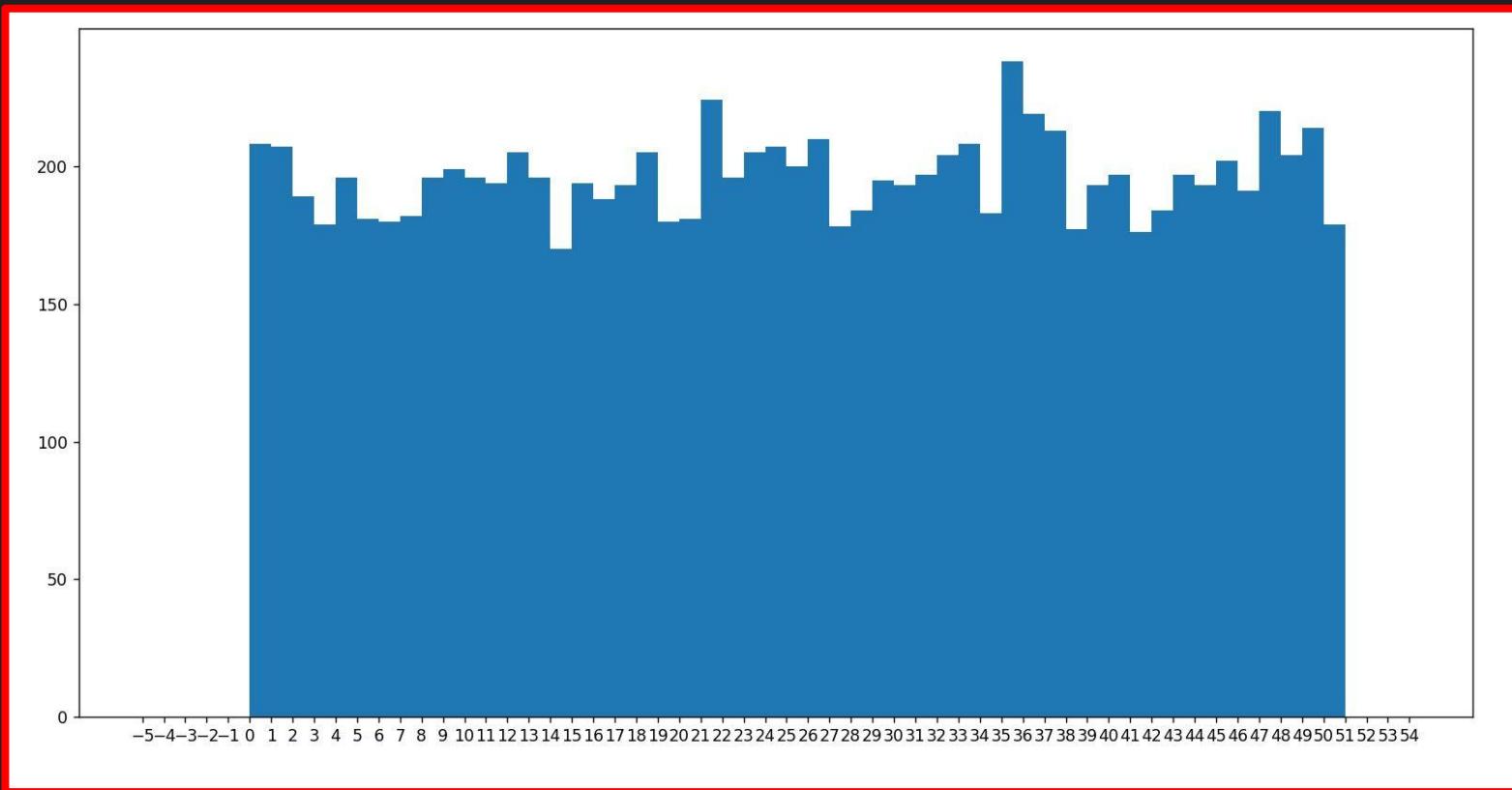
# uniform distribution
b = 51 # mean, max value of no of events per unit time
# a = 0 by default

# repeat experiment 10000 times.
rng = default_rng()
s = rng.integers(b, size=10000) #len(s) = 10000

myBins = np.arange(-5, 55, 1) # [-5 -4 ... . 1 2 ... 54]

plt.hist(s, bins = myBins)
plt.xticks(myBins)
plt.show()
```

Sampling data from Uniform Distribution



Continuous Random Variable

Random variables having value within a continuum are called continuous random variables.

- Examples: height, volume, vector's direction
- Probability that continuous random variable takes a particular value is zero.
- Why?
- Uniform distribution in range $[0, 4]$, each value is equally likely, and there are infinitely many values.
- Does zero probability mean that event will never occur?

Continuous Random Variable

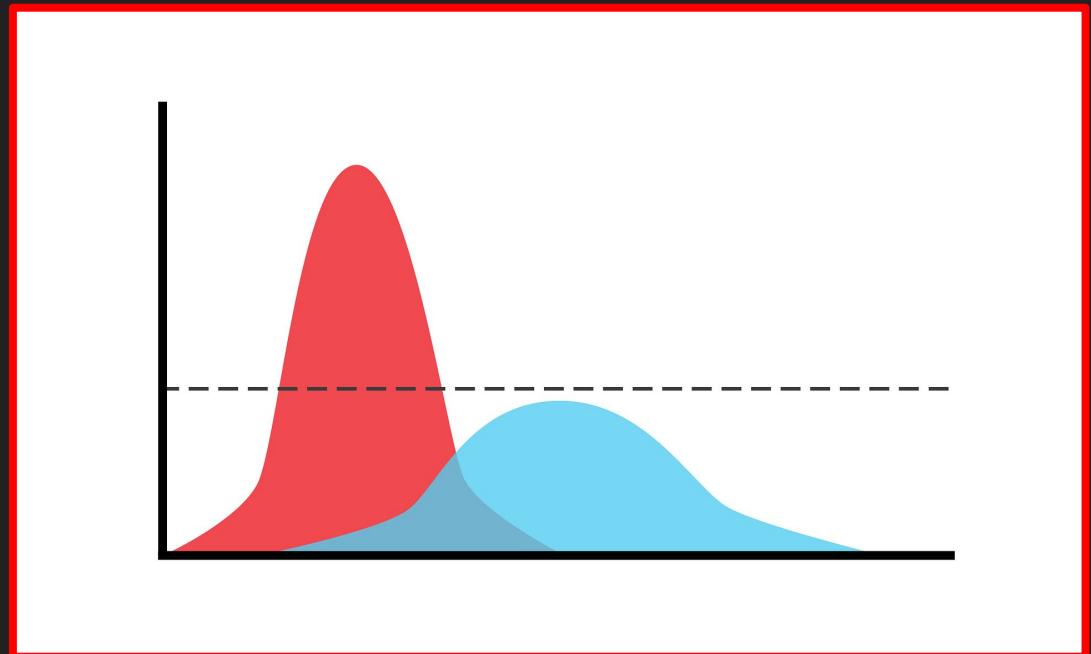
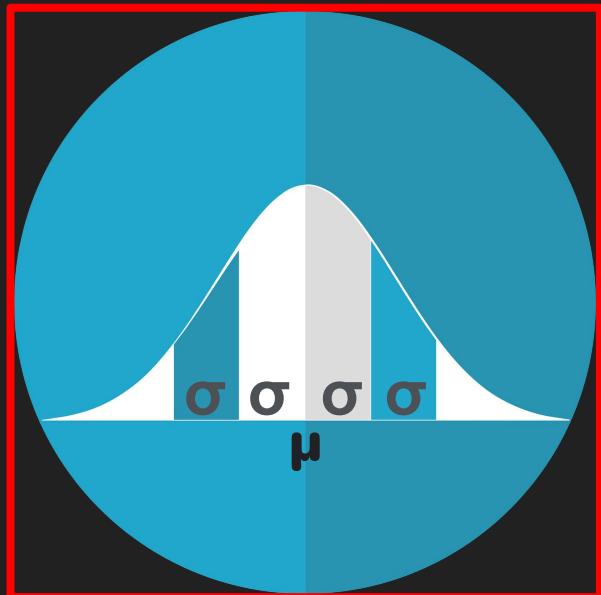
- Hence for continuous r.v. we consider cumulative distribution function (cdf):
- $F_x(x) = P\{X \leq x\}$
- cdf can be used to compute cumulative interval measures, i.e., Probability that X lies in range $(a, b]$:
- $P(a < X \leq b) = F_x(b) - F_x(a)$

Probability Density Function

- Probability density function (pdf) of random variable x is
- Derivative of its cdf at that value x , denoted as $f_X(x) = dF_X(x)/dx$
- $P\{x \in A\} = \int_A f_X(x) dx$, for any set A of real numbers.
- Properties:
 - $\int_R f_X(x) dx = 1$, where R is $(-\infty, \infty)$
 - $P(a \leq X \leq b) = \int_{[a,b]} f_X(x) dx = F_x(b) - F_x(a)$
 - $P(X = a) = \int_{[a,a]} f_X(x) dx = F_x(a) - F_x(a) = 0$

Popular pdfs: Gaussian

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$



Popular pdfs: Uniform

- $f_X(x) = (b-a)^{-1}$, $a \leq x \leq b$
0, otherwise



Expected Value of Continuous Random Variable

$$E(X) = \int_R x f_X(x) dx, \text{ where } R \text{ is } (-\infty, \infty)$$

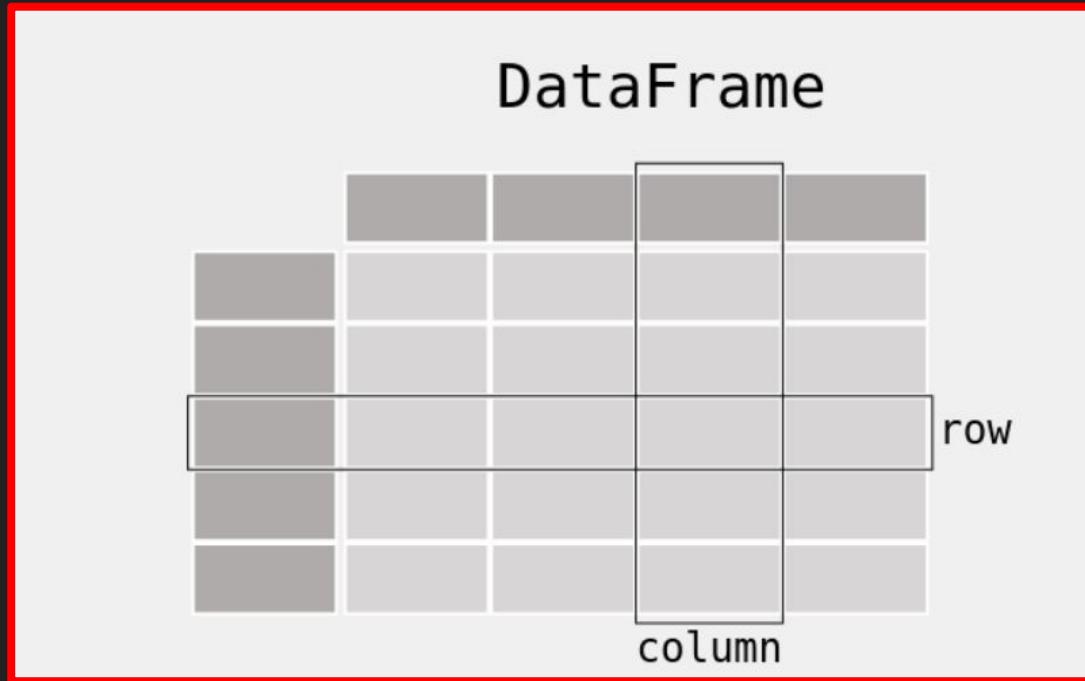
- Find, what are values of $E(X)$ for a gaussian random variable, and a uniform random variable?
- What is pdf for the Pareto distribution, given by:
- $f_X(x) = a (x_m)^a / x^{a+1}$ for $x \geq x_m$, 0 otherwise.
- a and x_m are the parameters of pareto dn. You can assume them to be constant.

Data manipulation with Pandas

Pandas

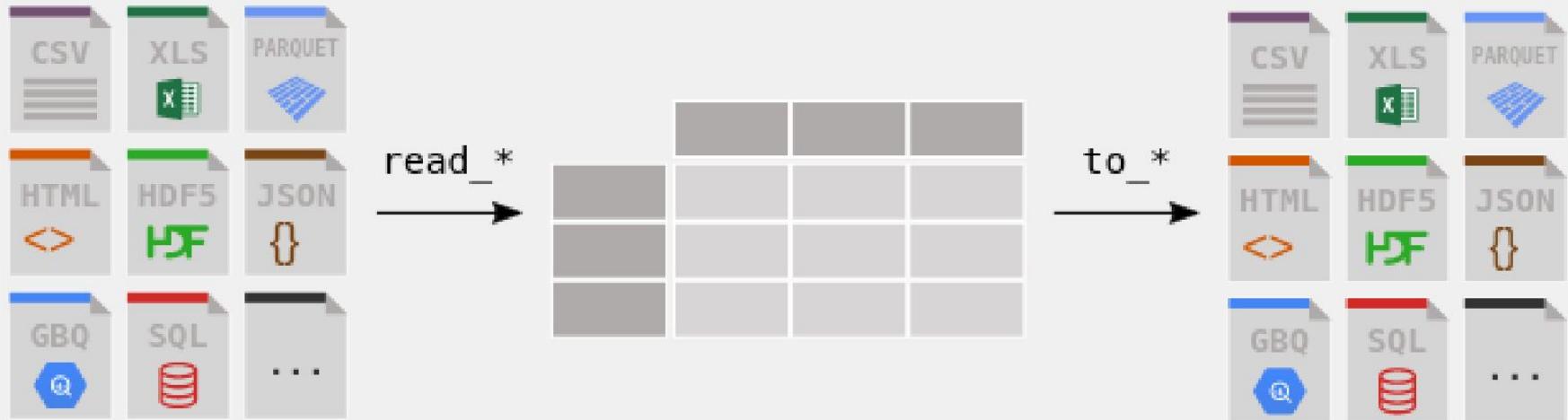
- Pandas builds on NumPy and ndarray
- DataFrame, Series: basic data structure provided by Pandas
- DataFrame is a multidimensional array, with attached row and column labels; Series is a 1-D array
- Heterogeneous types, and/or missing data
- More flexible than ndarrays: add labels, handle missing data, grouping data etc.

Idea of DataFrame



From <https://pandas.pydata.org/>

Pandas support reading/writing various formats



From <https://pandas.pydata.org/>

Pandas Series

- `class pandas.Series(data=None, index=None, dtype=None, name=None, ...)`
- One-dimensional ndarray with axis labels (including time series).
- Data: Contains data stored in Series. If data is a dict, argument order is maintained.
- Index: Values must have the same length as data. Non-unique index values are allowed. Will default to RangeIndex (0, 1, 2, ..., n) if not provided.
- dtype: Data type for the output Series. If not specified, this will be inferred from data.
- Namestr (optional): The name to give to the Series.

Pandas Series

```
>>> import pandas as pd  
  
>>> data = pd.Series([0.2, 0.4, 0.6, 0.8, 1])  
  
>>> print(data)  
  
0    0.2  
1    0.4  
2    0.6  
3    0.8  
4    1.0  
dtype: float64
```

Pandas Series

```
>>> import pandas as pd  
  
>>> data = pd.Series([0.2, 0.4, 0.6, 0.8, 1])  
  
>>> print(data)  
  
0    0.2  
1    0.4  
2    0.6  
3    0.8  
4    1.0  
dtype: float64
```

Pandas Series

```
>>> data.values  
array([0.2, 0.4, 0.6, 0.8, 1. ])  
>>> type(data.values)  
<class 'numpy.ndarray'>  
>>> data.index  
RangeIndex(start=0, stop=5, step=1)  
>>> data[1]  
0.4  
>>> data[1:3]  
1    0.4  
2    0.6  
dtype: float64
```

```
>>> import pandas as pd  
  
>>> data = pd.Series([0.2, 0.4, 0.6, 0.8, 1])  
  
>>> print(data)  
  
0    0.2  
1    0.4  
2    0.6  
3    0.8  
4    1.0  
dtype: float64
```

explicit index

Pandas Series (Similar to dict)

```
>>> population_dict = {'H.P.': 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
>>> population = pd.Series(population_dict)
>>> population.values
array([ 6800000., 28000000., 25000000.])
>>> population.index
Index(['H.P.', 'Punjab', 'Haryana'], dtype='object')
>>> population['H.P.']
6800000.0
>>> population['H.P.':'Haryana']
```

Pandas Series (Similar to dict)

```
>>> population_dict = {'H.P.': 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
>>> population = pd.Series(population_dict)
>>> population.values
array([ 6800000., 28000000., 25000000.])
>>> population.index
Index(['H.P.', 'Punjab', 'Haryana'], dtype='object')
>>> population['H.P.']
6800000.0
>>> population['H.P.':'Haryana']
H.P.      6800000.0
Punjab   28000000.0
Haryana  25000000.0
dtype: float64
```

Pandas Series

```
data3 = pd.Series({2:'a', 3:'b', 1:'c'}, index = [3 ,2])
```

```
3    b  
2    a  
dtype: object
```

DataFrame

- DataFrame is a 2-D array with flexible row indices and column names
- DF is a sequence of aligned Series objects
- Aligned = shares same index

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})
print(states)
```

DataFrame

- DataFrame is a 2-D array with flexible row indices and column names
- DF is a sequence of aligned Series objects
- Aligned = shares same index

```
import pandas as pd

population_dict = {'H.P': 6800000.0, 'Punjab': 28000000.0, 'Haryana': 25000000.0}
area_dict = {'H.P': 55673, 'Punjab': 50362, 'Haryana': 44212}
ana': 2.5e7}

population = pd.Series(population_dict)
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})
print(states)
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1

#Index(['H.P', 'Punjab', 'Haryana'], dtype='object')
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1
#Index(['H.P', 'Punjab', 'Haryana'], dtype='object')

print(states.columns) # attribute 2, DF has columns arritibute
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1
#Index(['H.P', 'Punjab', 'Haryana'], dtype='object')

print(states.columns) # attribute 2, DF has columns arritibute
#Index(['population', 'area'], dtype='object')
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1
#Index(['H.P', 'Punjab', 'Haryana'], dtype='object')

print(states.columns) # attribute 2, DF has columns arritibute
#Index(['population', 'area'], dtype='object')

print(states['area']) # DF maps a column name to a Series object
```

```
import pandas as pd

population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}
population = pd.Series(population_dict)

area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}
area = pd.Series(area_dict)

states = pd.DataFrame({'population': population, 'area': area})

print(states.index) # attribute 1
#Index(['H.P', 'Punjab', 'Haryana'], dtype='object')

print(states.columns) # attribute 2, DF has columns arritibute
#Index(['population', 'area'], dtype='object')

print(states['area']) # DF maps a column name to a Series object
...
H.P      55673
Punjab   50362
Haryana  44212
Name: area, dtype: int64
'''
```

Creating a DataFrame From a single Series object

```
# states = pd.DataFrame({'population': population, 'area': area})  
  
>>> import pandas as pd  
  
>>> df_pop1 = pd.DataFrame(population, columns = ['population'])  
  
>>> df_pop2 = pd.DataFrame({'population': population})
```

	Population
H.P	6800000.0
Punjab	28000000.0
Haryana	25000000.0

Creating a DataFrame From a List of Dictionaries

```
data = [ { 'a': i, 'b': 2*i } for i in range (3) ]
```

```
print(data)
```

```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]
```

```
df_data = pd.DataFrame(data) # dict keys forms the column names
```

```
print(df_data)
```

	a	b
0	0	0
1	1	2
2	2	4

Creating a DataFrame From 2D Numpy Array

```
>>> import pandas as pd  
  
>>> import numpy as np  
  
>>> df_random_data = pd.DataFrame(np.random.rand(4,3), columns  
= ['c1', 'c2', 'c3'], index = ['r1', 'r2', 'r3', 'r4'])  
  
>>> df_random_data
```

	c1	c2	c3
r1	0.282086	0.242101	0.563668
r2	0.213973	0.172746	0.674611
r3	0.725879	0.175356	0.559528
r4	0.576773	0.477594	0.059927

Handling Missing Values

```
>>> import pandas as pd  
>>> import numpy as np  
>>> df_dict_data = pd.DataFrame([{'a': 1, 'b': 2}, {'a': 1, 'c': 2}])  
>>> df_dict_data
```

	a	b	c
0	1	2.0	NaN
1	1	NaN	2.0

Changing/Appending Values and Explicit Indexing

```
>>> import pandas as pd  
>>> import numpy as np  
>>> data = pd.Series(np.arange(0.2,1.2,0.2), \  
 index = ['a', 'b', 'c', 'd', 'e'])  
>>> data['b':'d'] # 'd' is included in explicit indexing  
b    0.4  
c    0.6  
d    0.8  
dtype: float64
```

```
>>> data['b'] = 10 # changing  
value  
>>> data['f'] = 100 # appending  
>>> data  
a    0.2  
b    10.0  
c    0.6  
d    0.8  
e    1.0  
f    100.0  
dtype: float64
```

Integer Indices Can Be Confusing

```
>>> import pandas as pd
```

```
>>> data = pd.Series(['a', 'b', 'c'], index = [1, 2, 3])
```

```
>>> data
```

```
1  a  
2  b  
3  c  
dtype: object
```

```
>>> data[1] # explicit index used here
```

```
'a'
```

```
>>> data[0:1] # implicit index while slicing, may give warning
```

```
1  a  
dtype: object
```

```
>>> data.loc[1:2] # explicit index
```

```
1  a  
2  b  
dtype: object
```

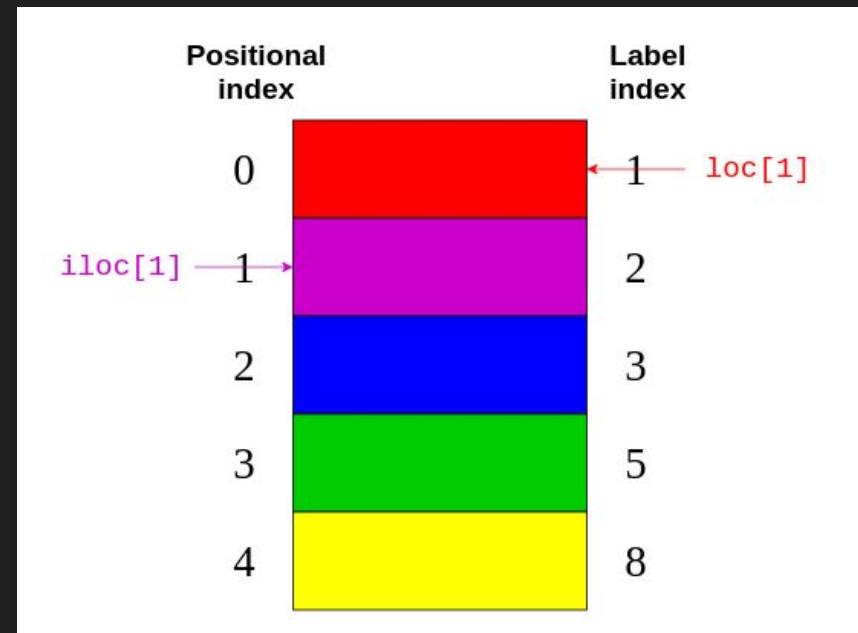
```
>>> data.iloc[1:2] # implicit index
```

```
2  b  
dtype: object
```

Integer Indices Can Be Confusing

```
>>> import pandas as pd
```

```
>>> colors = pd.Series(['red', 'purple', 'blue', 'green', 'yellow'], \  
index = [1, 2, 3, 5, 8])
```



Integer Indices Can Be Confusing

```
>>> import pandas as pd                                     >>> a = [1, 2, 3]
>>> data = pd.Series(['a', 'b', 'c'], index = [1, 2, 3])      >>> a[1:7]
>>> data.loc[1:7]                                         [2, 3]
1  a
2  b
3  c
dtype: object

>>> data.iloc[1:7]

2  b
3  c
dtype: object
```

String Indices

```
>>> import pandas as pd
```

```
>>> data = pd.Series(['a', 'b', 'c'], index = ['d', 'e', 'f'])
```

```
>>> data
```

```
d    a  
e    b  
f    c  
dtype: object
```

```
>>> data[1] # does not output 'a' like previous slide, resolves as implicit
```

```
'b'
```

```
>>> data[0:1] # implicit index
```

```
d    a  
dtype: object
```

```
>>> data['d':'f'] # resolves as explicit
```

```
d    a  
e    b  
f    c  
dtype: object
```

```
>>> # in explicit: last item is included
```

Adding New Columns In DataFrame

```
>>> import pandas as pd  
  
>>> population_dict = {'H.P' : 6.8e6, 'Punjab': 2.8e7, 'Haryana': 2.5e7}  
  
>>> population = pd.Series(population_dict)  
  
>>> area_dict = {'H.P' : 55673, 'Punjab': 50362, 'Haryana': 44212}  
  
>>> area = pd.Series(area_dict)  
  
>>> states = pd.DataFrame({'population': population, 'area': area})  
  
>>> states
```

	population	area
H.P	6800000.0	55673
Punjab	28000000.0	50362
Haryana	25000000.0	44212

```
>>> states['density'] = \  
states['population']/states['area']  
  
>>> states
```

	population	area	density
H.P	6800000.0	55673	122.1417
Punjab	28000000.0	50362	555.9747
Haryana	25000000.0	44212	565.4573

Accessing Values As ndarray In DataFrame

```
>>> states
```

	population	area	density
H.P	6800000.0	55673	122.141792
Punjab	28000000.0	50362	555.974743
Haryana	25000000.0	44212	565.457342

```
>>> states.values # Returns ndarray
```

```
array([[6.8e+06, 5.5673e+04, 1.22141792e+02],  
       [2.8e+07, 5.0362e+04, 5.55974743e+02],  
       [2.5e+07, 4.4212e+04, 5.65457342e+02]])
```

```
>>> states.values[0] # Returns a row
```

```
array([6.8e+06, 5.5673e+04, 1.22141792e+02])
```

Indexers in DataFrame

```
>>> states
```

```
          population      area      density
H.P      6800000.0    55673  122.141792
Punjab  28000000.0   50362  555.974743
Haryana 25000000.0   44212  565.457342
```

```
>>> states.iloc[:2, :2] # iloc indexer
```

```
          population      area
H.P      6800000.0    55673
Punjab  28000000.0   50362
```

```
>>> states.loc[:'Haryana', :'area'] # loc indexer
```

```
          population      area
H.P      6800000.0    55673
Punjab  28000000.0   50362
Haryana 25000000.0   44212
```

Indices and Columns Preserved while applying Ufuncs

```
>>> rng = np.random.RandomState(seed = 42)
```

```
>>> ser = pd.Series(rng.randint(0, 10, 4))
```

```
>>> ser
```

```
0    6  
1    3  
2    7  
3    4  
dtype: int32
```

```
>>> np.exp(ser)
```

```
0      403.428793  
1      20.085537  
2     1096.633158  
3      54.598150  
dtype: float64
```

Indices and Columns Preserved while applying Ufuncs

```
>>> rng = np.random.RandomState(seed = 42)
```

```
>>> df = pd.DataFrame(rng.randint(0, 10, (2,3)))
```

```
>>> df
```

```
 0 1 2  
0 6 9 2  
1 6 7 4
```

```
>>> np.sin(df * np.pi / 4)
```

```
      0          1          2  
0 -1.0  0.707107  1.000000e+00  
1 -1.0 -0.707107  1.224647e-16
```

Index Alignment for Series

```
>>> A = pd.Series([2, 5, 7], index = [0, 1, 2])
```

```
>>> B = pd.Series([1, 3, 5], index = [1, 2, 3])
```

```
>>> A + B
```

```
0    NaN
```

```
1    6.0
```

```
2   10.0
```

```
3    NaN
```

```
dtype: float64
```

Index Alignment for Series

```
>>> A = pd.DataFrame(np.random.randint(0, 20, (2,2)), columns = list('ab'))  
>>> B = pd.DataFrame(np.random.randint(0, 10, (3,3)), columns = list('bac'))  
>>> A + B  
      a   b   c  
0  20.0  8.0  NaN  
1  18.0 22.0  NaN  
2  NaN   NaN  NaN  
  
>>> fill = A.stack().mean()  
  
>>> fill  
11.25  
  
>>> A.add(B, fill_value = fill)  
      a     b     c  
0  20.00  8.00 15.25  
1  18.00 22.00 13.25  
2  14.25 20.25 15.25  
  
>>> A  
      a   b  
0  13   2  
1  13  17  
  
>>> B  
      b   a   c  
0   6   7   4  
1   5   5   2  
2   9   3   4
```

Reading titanic.csv

```
>>> df = pd.read_csv('titanic.csv')  
  
>>> type(df)  
  
<class 'pandas.core.frame.DataFrame'>  
  
>>> df
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
..
888	889	0	3	...	23.4500	NaN	S
889	890	1	1	...	30.0000	C148	C
890	891	0	3	...	7.7500	NaN	Q

[891 rows x 12 columns]

Most of this material is from pandas.pydata.org/docs

Top n rows

```
>>> df.head()
```

```
   PassengerId  Survived  Pclass ...    Fare  Cabin  Embarked
0            1         0      3 ...  7.2500    NaN       S
1            2         1      1 ... 71.2833    C85       C
2            3         1      3 ...  7.9250    NaN       S
3            4         1      1 ... 53.1000    C123       S
4            5         0      3 ...  8.0500    NaN       S
```

[5 rows x 12 columns]

```
>>> df.head(2)
```

```
   PassengerId  Survived  Pclass ...    Fare  Cabin  Embarked
0            1         0      3 ...  7.2500    NaN       S
1            2         1      1 ... 71.2833    C85       C
```

[2 rows x 12 columns]

Last n rows

```
>>> df.tail(3)
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
888	889	0	3	...	23.4500	NaN	S
889	890	1	1	...	30.0000	C148	C
890	891	0	3	...	7.7500	NaN	Q

[3 rows x 12 columns]

```
>>> df.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Data Types in a Data Frame

```
>>> df.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name             object
Sex              object
Age              float64
SibSp            int64
Parch            int64
Ticket           object
Fare             float64
Cabin            object
Embarked         object
dtype='object'
```

Generate Descriptive Statistics

```
>>> df.describe()
```

	PassengerId	Survived	Pclass	...	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	...	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	...	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	...	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	...	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	...	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	...	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	...	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	...	8.000000	6.000000	512.329200

[8 rows x 7 columns]

Select Specific Columns

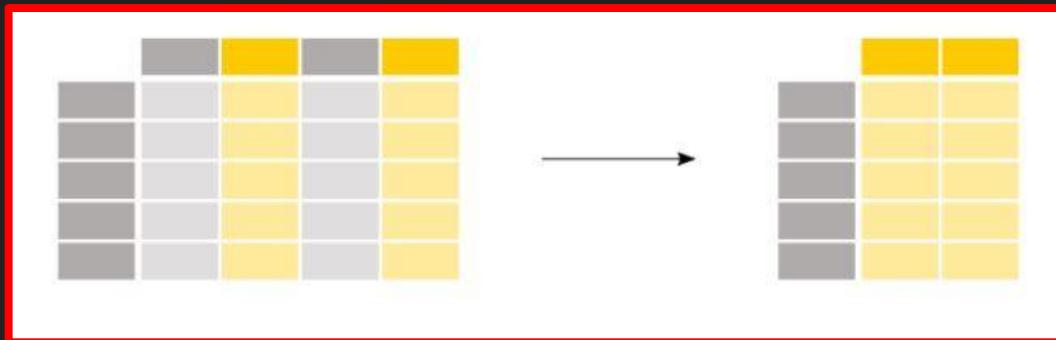
```
>>> df1 = df[['Age', 'Sex']]
```

```
>>> df1
```

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male

.. ...
886 27.0 male
887 19.0 female
888 NaN female
889 26.0 male
890 32.0 male

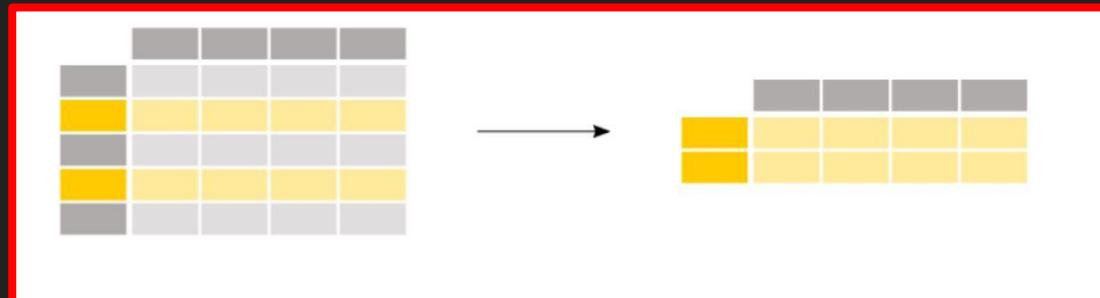
[891 rows x 2 columns]



Select Specific Rows

```
>>> df['Age'] > 35
```

```
0    False  
1    True  
2   False  
3   False  
4   False  
...  
886  False  
887  False  
888  False  
889  False  
890  False  
  
Name: Age, Length: 891, dtype: bool
```



```
>>> above35df = df[df['Age'] > 35]
```

```
>>> above35df.shape
```

```
(217, 12)
```

```
>>> df.shape
```

```
(891, 12)
```

Select Specific Rows

```
>>> class2Or3 = df[df['Pclass'].isin([2,3])] #Pclass is Passenger's class
```

```
>>> class2Or3
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked	
0		1	0	3	...	7.2500	NaN	S
2		3	1	3	...	7.9250	NaN	S
4		5	0	3	...	8.0500	NaN	S
5		6	0	3	...	8.4583	NaN	Q
7		8	0	3	...	21.0750	NaN	S
..
884	885	0	3	...	7.0500	NaN	S	
885	886	0	3	...	29.1250	NaN	Q	
886	887	0	2	...	13.0000	NaN	S	
888	889	0	3	...	23.4500	NaN	S	
890	891	0	3	...	7.7500	NaN	Q	

[675 rows x 12 columns]

Select Specific Rows

```
>>> class2Or3 = df[(df['Pclass'] == 2) | (df['Pclass'] == 3)]
```

```
>>> class2Or3
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
2	3	1	3	...	7.9250	NaN	S
4	5	0	3	...	8.0500	NaN	S
5	6	0	3	...	8.4583	NaN	Q
7	8	0	3	...	21.0750	NaN	S
..
884	885	0	3	...	7.0500	NaN	S
885	886	0	3	...	29.1250	NaN	Q
886	887	0	2	...	13.0000	NaN	S
888	889	0	3	...	23.4500	NaN	S
890	891	0	3	...	7.7500	NaN	Q

[675 rows x 12 columns]

Choose Specific Rows and Columns

```
>>> # names of passengers older than 35 years
```

```
>>> df.loc[df['Age'] > 35, 'Name'] #or i) temp = df[df['Age'] > 35], ii) temp['Name']
```

```
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  
6
```

```
           McCarthy, Mr. Timothy J
```

```
11          Bonnell, Miss. Elizabeth
```

```
13          Andersson, Mr. Anders Johan
```

```
15          Hewlett, Mrs. (Mary D Kingcome)
```

```
...
```

```
865          Bystrom, Mrs. (Karolina)
```

```
871          Beckwith, Mrs. Richard Leonard (Sallie Monopeny)
```

```
873          Vander Cruyssen, Mr. Victor
```

```
879          Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)
```

```
885          Rice, Mrs. William (Margaret Norton)
```

```
Name: Name, Length: 217, dtype: object
```

Choose Specific Rows and Columns

```
>>> # names of passengers older than 35 years
```

```
>>> df.loc[10:15, [ 'PassengerId', 'Pclass', 'Name']] # label indexing with loc
```

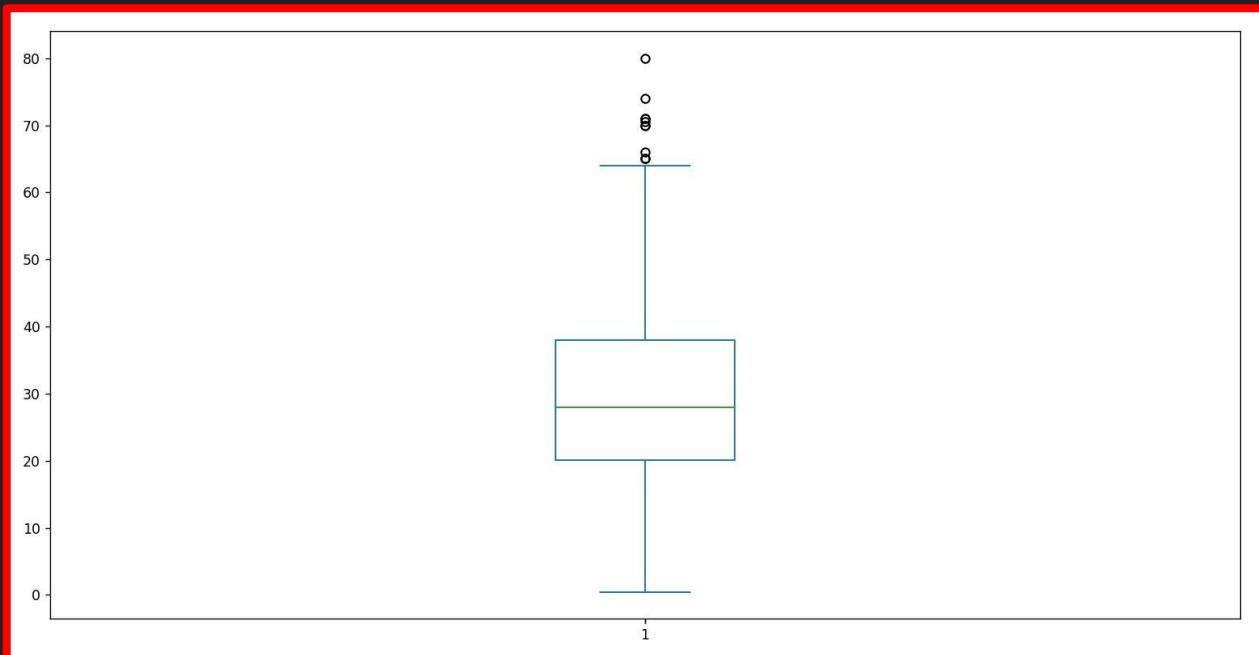
	PassengerId	Pclass	Name
10	11	3	Sandstrom, Miss. Marguerite Rut
11	12	1	Bonnell, Miss. Elizabeth
12	13	3	Saundercock, Mr. William Henry
13	14	3	Andersson, Mr. Anders Johan
14	15	3	Vestrom, Miss. Hulda Amanda Adolfina
15	16	2	Hewlett, Mrs. (Mary D Kingcome)

Pandas supports plotting, based on Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> df['Age'].plot.box()
```

```
>>> plt.show()
```



Aggregation

```
>>> df['Age'].min()
```

0.42

```
>>> df['Age'].max()
```

80.0

```
>>> df['Age'].value_counts()
```

24.00 30

22.00 27

18.00 26

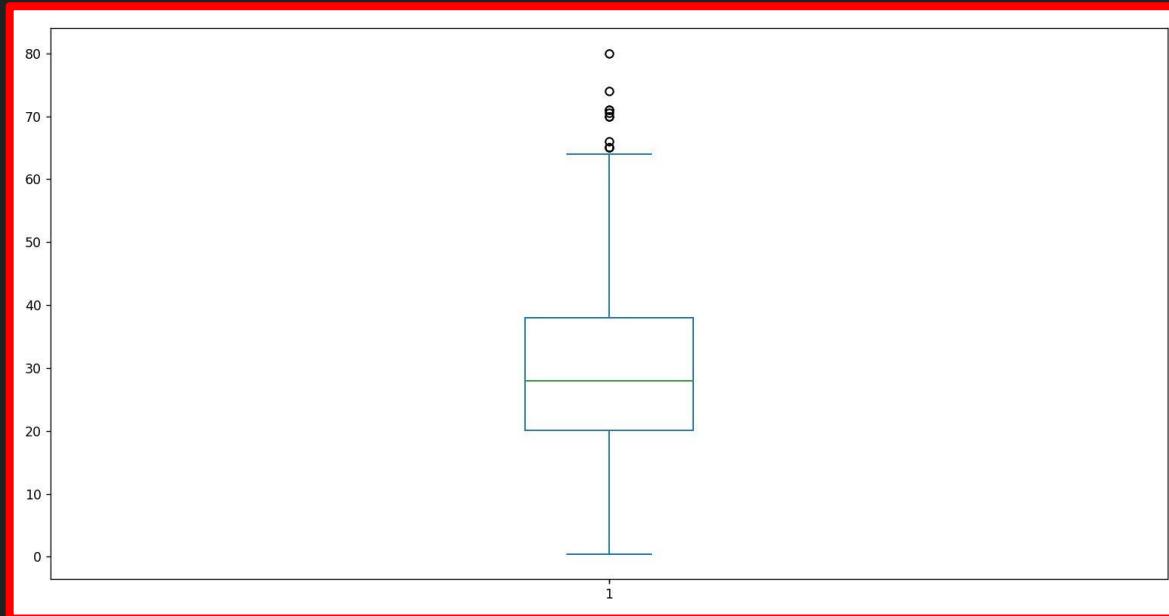
..

0.92 1

23.50 1

74.00 1

Name: Age, Length: 88, dtype: int64



Aggregation: Grouping

```
>>> df.groupby('Pclass')['Age'].mean()
```

Pclass

1	38.233441
2	29.877630
3	25.140620

Name: Age, dtype: float64

```
>>> df[df['Pclass'] == 1]['Age'].mean()
```

38.233440860215055

Aggregation: Grouping

```
>>> df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          -----          ----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin         204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Aggregation: Grouping

- `df.dropna()`
 - Drop a row having missing values
- `df.dropna(axis=1)`
 - Drop a column having missing values
- `df.fillna()`
 - Replace nan with some default value

Aggregation: Grouping

```
>>> dfCpy = df.copy()
```

```
>>> dfCpy['Cabin'].fillna(value=0\  
, inplace = True)
```

```
>>> dfCpy.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
 #   Column            Non-Null Count  Dtype     
 ---  --     
 0   PassengerId      891 non-null    int64    
 1   Survived         891 non-null    int64    
 2   Pclass           891 non-null    int64    
 3   Name             891 non-null    object    
 4   Sex              891 non-null    object    
 5   Age              714 non-null    float64   
 6   SibSp            891 non-null    int64    
 7   Parch            891 non-null    int64    
 8   Ticket           891 non-null    object    
 9   Fare             891 non-null    float64   
 10  Cabin            891 non-null    object    
 11  Embarked         889 non-null    object    
 dtypes: float64(2), int64(5), object(5)  
 memory usage: 83.7+ KB
```

Aggregation: Grouping

```
>>> nba = pd.read_csv('nbaallello.csv')
```

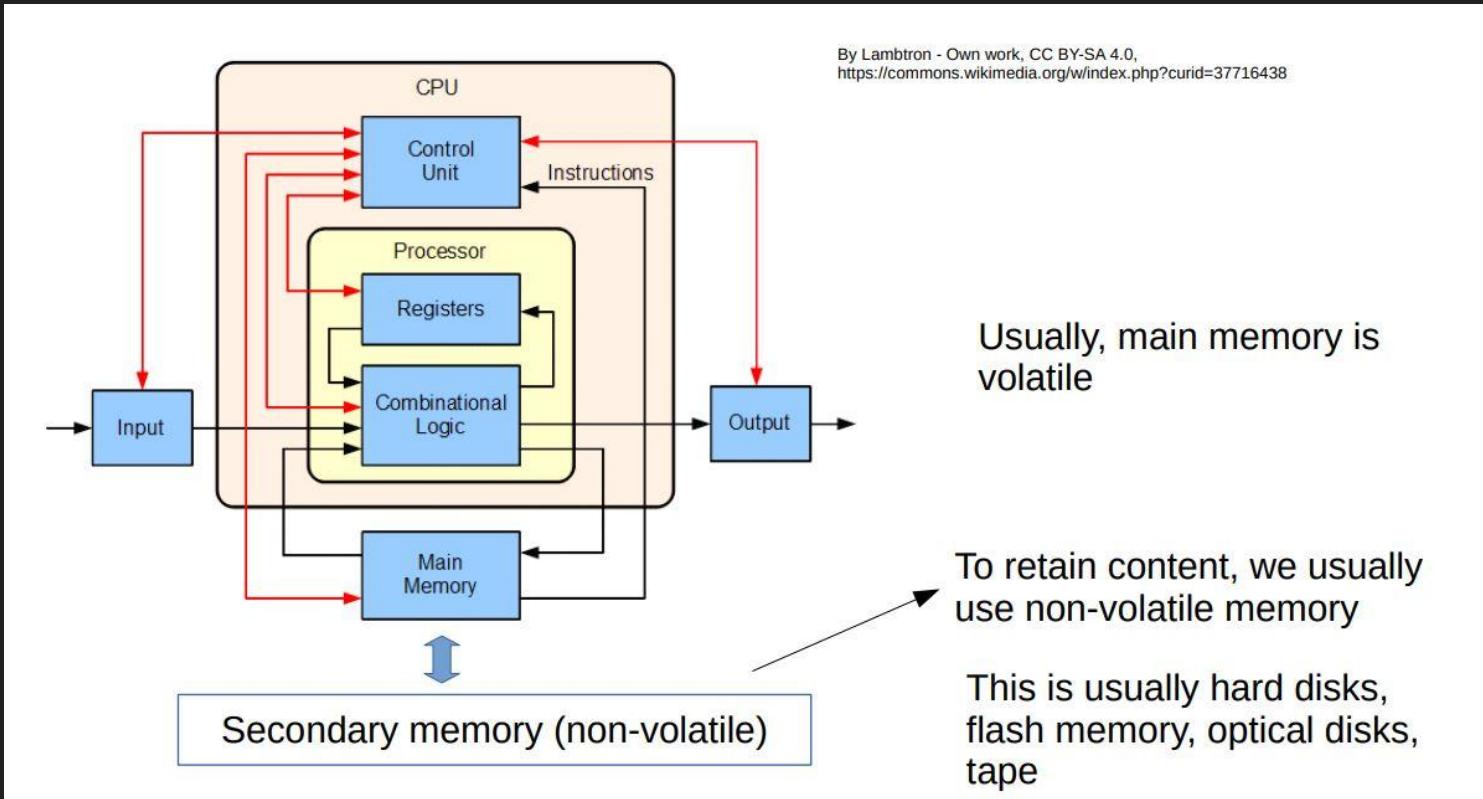
```
>>> nba[(nba['fran_id'] == 'Spurs') & (nba['year_id'] > 2010)].groupby(['year_id', 'game_result'])['game_id'].count()
```

year_id	game_result	game_id count
2011	L	25
	W	63
2012	L	20
	W	60
2013	L	30
	W	73
2014	L	27
	W	78
2015	L	31
	W	58

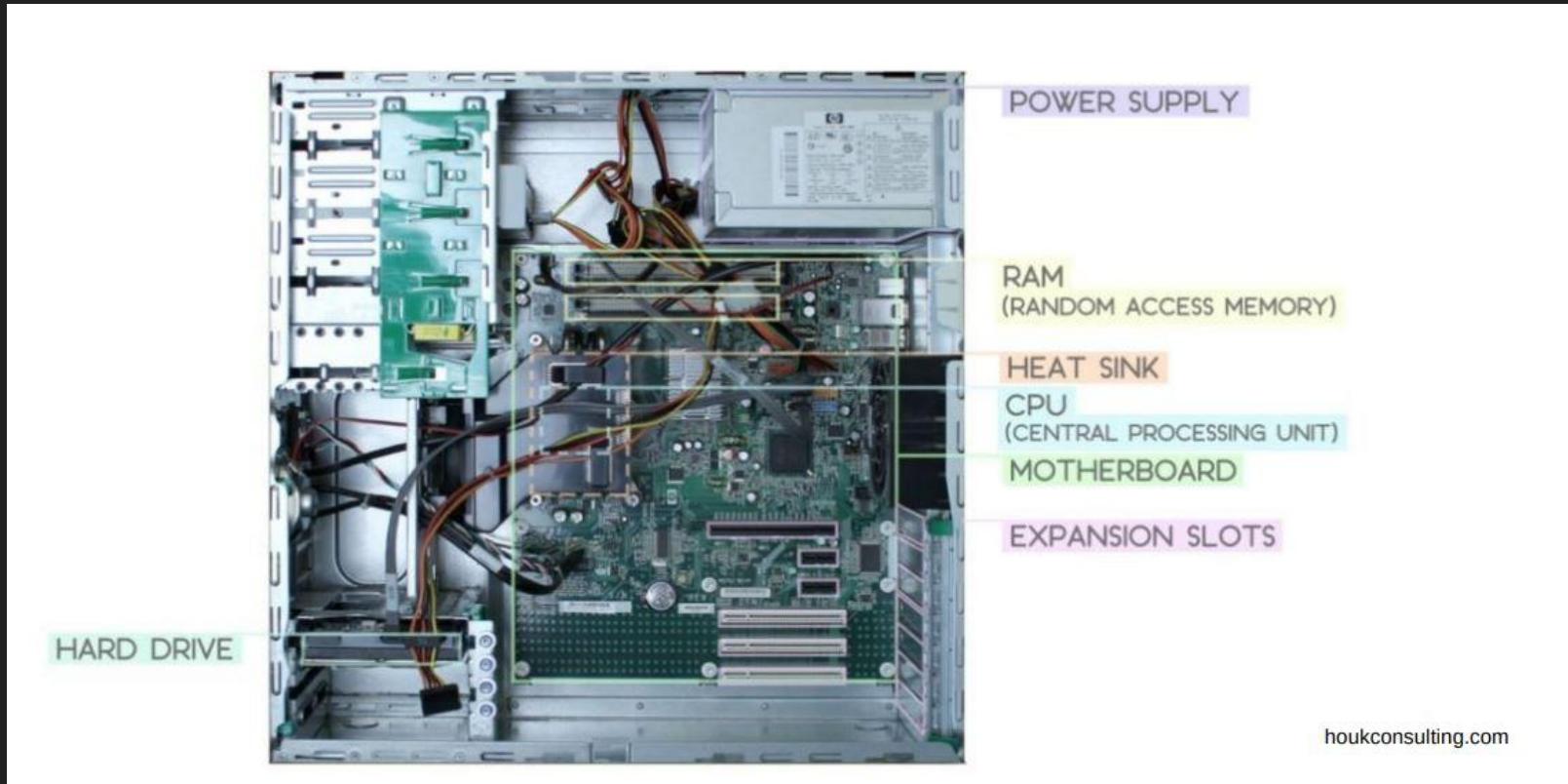
```
Name: game_id, dtype: int64
```

Input and Output

Parts of Computer



Parts of Computer



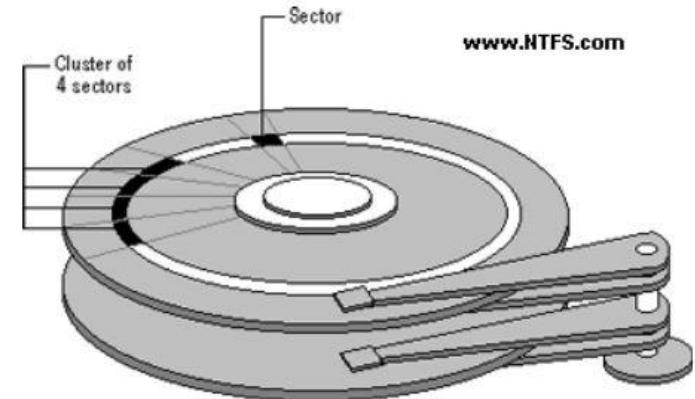
Inside a Hard Disk Drive (HDD)



<https://www.nextofwindows.com/>

For more insights, watch this video:

<https://www.youtube.com/watch?v=NtPc0jl21i0>



Made up of tracks and sectors

Solid State Device (SSD)



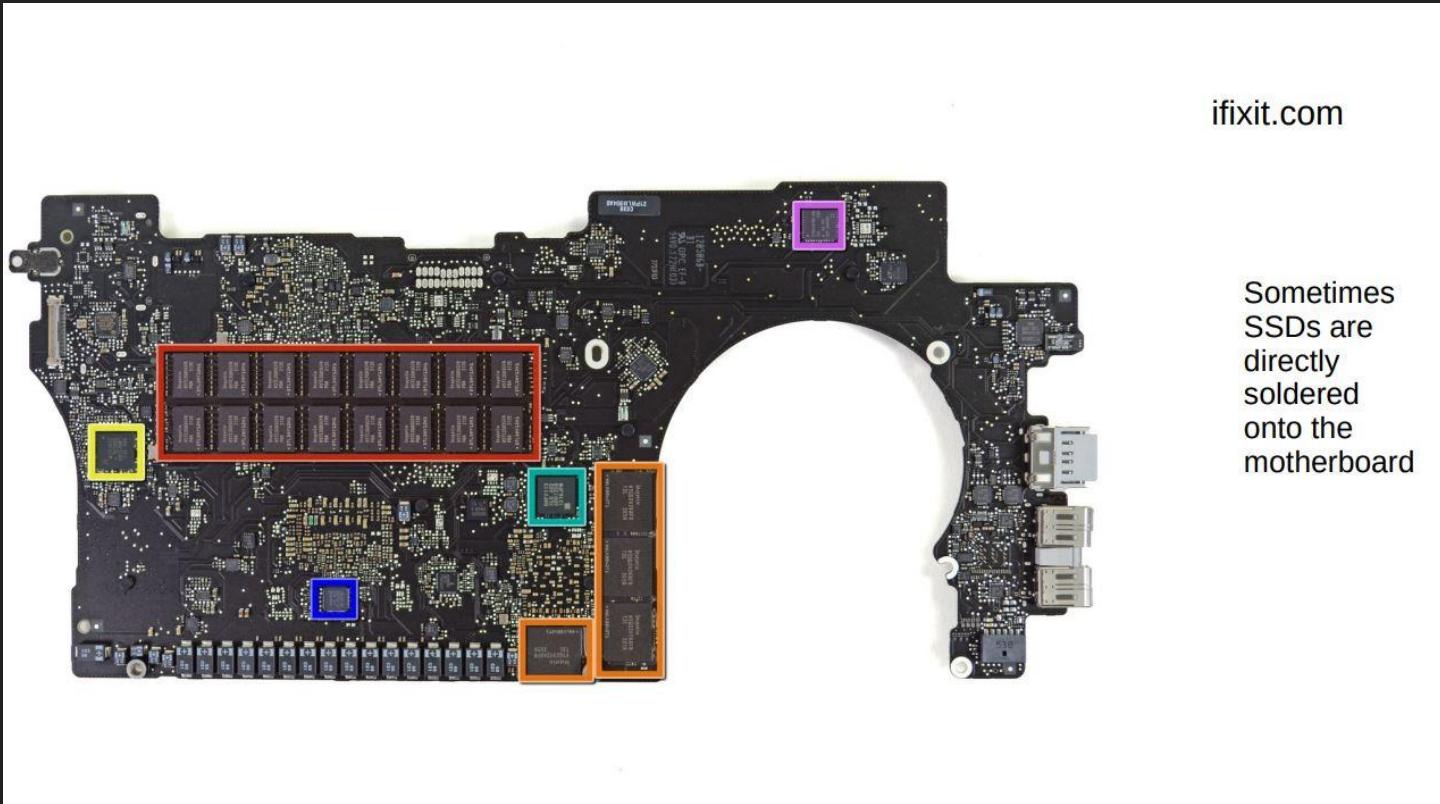
https://commons.wikimedia.org/wiki/File:Samsung_SSD_840_120GB_MZ-7TD120-4_LID_REMOVED.JPG

Solid-state drives are more reliable than HDDs
SSDs are becoming mainstream in many computers



https://commons.wikimedia.org/wiki/File:Super_Talent_2.5in_SATA_SSD_SAM64GM25S.jpg

Solid State Device (SSD)



Why Input Output is Challenging?

Vast speed difference between CPU+memory and electromechanical I/O devices

DVD	Laser	1 or 2 sided disk	600-1600 rpm	1-10 GB	ms-sec
Hard disk	Magnetic	2 to 30 sided disk	3600-15000 rpm	1 GB- 10 TB	ms
Solid state drive	Flash memory	Electronic	No mechanical movement	1 GB- 100 GB	μs-ms
Tape	Magnetic	500-900m reel	10 meters/sec	1 GB-300 PB	secs-hours
Cloud storage	Behind the network				ms-hours

Why Input Output (I/O) is Challenging?

- Vast speed difference between CPU+memory and electromechanical I/O devices
- The operating system hides these differences
- A file is provided as an abstraction for the physical device
- All I/O is performed with streams
 - Stream: Sequence of bytes in the OS memory

Why Input Output (I/O) is Challenging?

- A file is provided as an abstraction for the physical device
- Various types of files:
 - Binary file: named sequence of bytes
 - Text file (ASCII file):
 - Each byte is a character (see ASCII code, next slide)
 - Each line ends with '\n'
 - Formatted file
 - eg. tabular data (rows and columns)
 - .csv file: comma separated values
 - .ods, .xls: binary spreadsheet files
 - Semi-structured file: eg. a web page (HTML file)

ASCII Code

ASCII (1977/1986)																	
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F	
0_0	NUL 0000	SOH 0001	STX 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F	
1_16	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F	
2_32	SP 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	*	+\br/>002B	,	- 002D	. 002E	/ 002F	
3_48	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	:	; 003A	< 003B	= 003C	> 003D	? 003E	003F
4_64	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F	
5_80	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	- 005F	
6_96	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F	
7_112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	} 007D	~ 007E	DEL 007F	

Hex code

Letter
 Number
 Punctuation
 Symbol
 Other
 Undefined
 Character changed from 1963 version and/or 1965 draft

HEX Code

- Values from right to left are multiples of 16
- `>>> ord('a')`
- 97
- Hex code of 'a' on previous slide is 0061, i.e. $1 * 16^0 + 6 * 16^1 = 97$
- `>>> ord('A')`
- 65
- Hex code of 'A' on previous slide is 0041, i.e. $1 * 16^0 + 4 * 16^1 = 65$
- A in 004A means 10, E in 004E means 14
- A: 10
- B: 11
- .
- .
- F: 15

ASCII Files

```
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ gedit myASCIIFile
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ cat myASCIIFile
this is an example of ASCII file.
there are 3 lines.
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ █
```

- CSV (Comma Separated Values) Files are text (ASCII) Files

```
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ cat myCSVFILE.csv
name, city, state
Rahul, Mandi, H.P
Kanika, Sonipat, Haryana
Vikram, Hyderabad, Telangana
```

Binary Files

- Binary files cannot be interpreted with ASCII codes
- An application (eg. an image viewer) is required to make sense of the contents

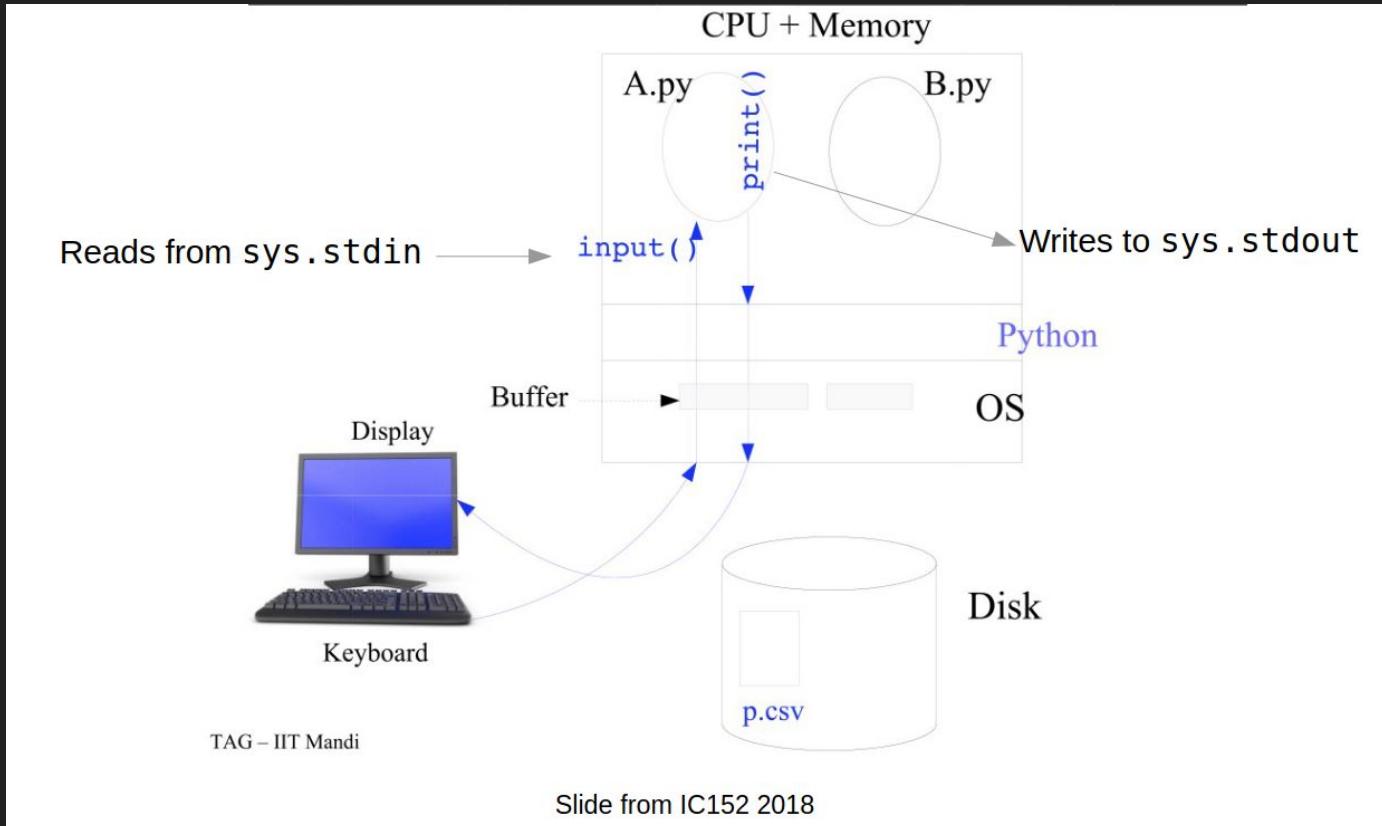
Binary Files

```
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ hexdump -C csv.png
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52 | .PNG.....IHDR|
00000010  00 00 07 03 00 00 00 a7  08 06 00 00 00 fd 9e f3 | .....|
00000020  c4 00 00 00 04 73 42 49  54 08 08 08 08 7c 08 64 | .....sBIT....|.d|
00000030  88 00 00 00 19 74 45 58  74 53 6f 66 74 77 61 72 | .....tEXtSoftwar|
00000040  65 00 67 6e 6f 6d 65 2d  73 63 72 65 65 6e 73 68 | e.gnome-screensh|
00000050  6f 74 ef 03 bf 3e 00 00  20 00 49 44 41 54 78 9c | ot...>... .IDATx.|
00000060  ec 9d 77 7c 5c 57 95 f8  bf f7 be e9 33 1a f5 66 | ..w|\W.....3..f|
00000070  59 ae 92 bb e3 de 5b e2  14 a7 3a cd 24 24 01 12 | Y.....[....$.$.|
00000080  42 a8 01 16 7e 0b 09 4b  db 85 cd 02 61 d9 4d 60 | B...~..K....a.M`|
00000090  0b 0b 2c 4b 08 10 52 48  ef 8e 63 27 b6 e3 b8 c4 | .,K..RH..c'....|
000000a0  49 dc 7b 95 2d 5b bd 6b  34 e5 bd fb fb 63 46 d2 | I.{.-[.k4....cF.|
000000b0  9b d1 48 1e 49 23 5b b6  e7 fb f9 f8 63 4d bb ef | ..H.I#[.....cM..|
000000c0  be 7b ee 3d f7 be 73 ee  39 57 cc 2f 9a a8 94 32 | .{.=..s.9W./...2|
000000d0  f0 d5 07 49 91 22 45 8a  14 29 52 a4 48 91 22 45 | ...I.."E..)R.H."E|
000000e0  8a 14 29 52 a4 48 91 22  45 8a 14 29 52 a4 48 91 | ..)R.H."E..)R.H.|
000000f0  22 c5 85 83 14 00 42 9c  eb 7a a4 48 91 22 45 8a | "...B..z.H."E.|
00000100  14 29 52 a4 48 91 22 45  8a 14 29 52 a4 48 91 22 | ..)R.H."E..)R.H.|
00000110  45 8a 14 29 52 a4 48 91  22 c9 58 14 02 a5 ce 75 | E..)R.H."X....u|
00000120  35 ce 22 79 8d 14 3c 58  8f 4d 0b bf 54 65 19 9c | 5."y..<X.M..Te..|
00000130  7c 14 45 10 20 b7 45 b5  20 40 10 25 45 00 14 20 | K.0.....T.0E. \
```

Standard Streams in Python

- Standard input stream: `sys.stdin`
 - Normally connected to the keyboard
- Standard output stream: `sys.stdout`
 - Normally connected to the display
- Standard error stream: `sys.stderr`
 - Normally connected to the display

Standard Streams in Python



I/O Redirection

- Standard input stream: sys.stdin
- I/O redirection can be used to connect stdin or stdout to a file, instead of keyboard or display
- This makes an interactive program into a non-interactive program

```
x = int(input())
y = int(input())
print(x+y)
```

I/O Redirection

```
x = int(input())
y = int(input())
print(x+y)
```

```
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ python3 addinputs.py
2
6
8
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ cat input
16
17
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ python3 addinputs.py < input
33
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ python3 addinputs.py < input > output
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ cat output
33
rohit@rohit-HP-Laptop-15s-fr2xxx:/media/rohit/04D8B01AD8B00C44/0000Mandi/Lec28Scripts$ █
```

File Access

- Basic steps:
 - Open the file
 - Read/write the file
 - Close the file

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

File Access

- `fp = open(filename,mode)`
- Different types of mode:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

<https://docs.python.org/3/library/functions.html#open>

File Access

- Basic steps:
- When done with a file, it must be closed using `f.close()`
- Else you may lose data!
- Buffering of output may happen without data actually being written to disk
- An exception or other issue (e.g., power failure) can happen

Other Methods in file Object

- `f.read()`: reads the entire file
- `f.read(n)`: reads n bytes of data from file
- `f.readline()`: reads one line
- `f.readlines()`: reads all lines
- `f.write(str)`: writes string str to file

Using Methods in file Object

```
# copyFile
# copies the content of one file to another
# written by user name on dd/mm/yyyy

def copyFileData(fSrc, fDst):
    while True:
        text = fSrc.read(50) # reads 50 bytes from the file.
        # Empty string means End of File (EOF)
        if text == "":
            break # to exit the loop
        fDst.write(text)

infilename = 'myFile.txt'
outfilename = 'myFileCopy.txt'

inFile = open(infilename, 'r')
outfile = open(outfilename, 'w')

copyFileData(inFile, outfile)

inFile.close()
outfile.close()
```

read(n) reads n bytes of data from the file.

Using Methods in file Object

```
# copyFileWoComments
# copies the content of one python file to another without copying comments
# written by user name on dd/mm/yyyy

def copyFileWoComments(fSrc, fDst):
    while True:
        text = fSrc.readline() # reads 50 bytes from the file.
        # Empty string means End of File (EOF)
        if text == "":
            break # to exit the loop
        if text[0] == '#':
            continue # continue to next iteration of loop from this point
            # leaves remaining part below
        fDst.write(text)
    break

inFilename = 'myFile.py'
outFilename = 'myFileCopy.py'

inFile = open(inFilename, 'r')
outFile = open(outFilename, 'w')

copyFileWoComments(inFile, outFile)

inFile.close()
outFile.close()
```

Using Methods in file Object

```
def copyFileWoComments(fSrc, fDst):
    while True:
        text = fSrc.readline() # reads 50 bytes from the file.
        # Empty string means End of File (EOF)
        if text == "":
            break # to exit the loop
        if text[0] == '#':
            continue # continue to next iteration of loop from this point
            # leaves remaining part below
        fDst.write(text)
    break
```

Using Methods in file Object

```
def copyFileWoComments(fSrc, fDst):
    while True:
        text = fSrc.readline() # reads 50 bytes from the file.
        # Empty string means End of File (EOF)
        if text == "":
            break # to exit the loop
        if text[0] == '#':
            continue # continue to next iteration of loop from this point
            # leaves remaining part below
        fDst.write(text)
    break
```

Command Line Arguments

```
# copyArgFileWoComments
# copies the content of one python file to another without copying comments
# Source and Destination Filenames are taken from arguments
# written by user name on dd/mm/yyyy

import sys

# sys.argv is a list
# check if arguments are passed, else exit with usage message
if len(sys.argv) <= 1:
    print('Usage: copyFileWoComments.py sourceFileName destFileName')
    sys.exit(-1)

#print(sys.argv[0]) # prints copyArgFileWoComments.py
inFilename = sys.argv[1]
outFilename = sys.argv[2]

inFile = open(inFilename, 'r')
outFile = open(outFilename, 'w')

copyFileWoComments(inFile, outFile)

inFile.close()
outFile.close()
```

```
def copyFileWoComments(fSrc, fDst):
    while True:
        text = fSrc.readline() # reads 50 bytes from the file.
        # Empty string means End of File (EOF)
        if text == "":
            break # to exit the loop
        if text[0] == '#':
            continue # continue to next iteration of loop from this point
            # leaves remaining part below
        fDst.write(text)
    break
```

Handling Exceptions

- Example of Exception:
- `f.open('missingFile.txt')` when missingFile.txt does not exist
- File not found
- No Permission to write, e.g. in C Drive
- Disk full while writing a file
- etc.

Handling Exceptions

```
C:\Users\Rohit\Lec29Scripts>python3 copyArgFileWoComments.py missingfile.txt missingfilecopy.txt
```

Traceback (most recent call last):

```
  File "C:\Users\Rohit\Lec29Scripts\copyArgFileWoComments.py", line 29, in <module>
```

```
    inFile = open(infilename, 'r')
```

```
FileNotFoundException: [Errno 2] No such file or directory: 'missingfile.txt'
```

```
C:\Users\Rohit\Lec29Scripts>
```

Handling Exceptions

```
import sys
import traceback

# handling exception using try except
# sys.argv is a list
# check if arguments are passed, else exit with usage message
if len(sys.argv) <= 1:
    print('Usage: copyFileWoComments.py sourceFileName destFileName')
    sys.exit(-1)

#print(sys.argv[0]) # prints copyArgFileWoComments.py
inFilename = sys.argv[1]
outFilename = sys.argv[2]

try:
    inFile = open(inFilename, 'r')
    outFile = open(outFilename, 'w')
except IOError:
    print('Error performing file IO')
    traceback.print_exc()
    sys.exit(-1)
```

Handling Exceptions

```
C:\Users\Rohit\Lec29Scripts>python3 handleError.py missingFile.txt missingFileCopy.txt
```

Error performing file IO

Traceback (most recent call last):

```
  File "C:\Users\Rohit\Lec29Scripts\handleError.py", line 16, in <module>
    inFile = open(inFilename, 'r')
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'missingFile.txt'
```

Handling Exceptions

```
import sys
import traceback

# handling exception using try except
# sys.argv is a list
# check if arguments are passed, else exit with usage message
if len(sys.argv) <= 1:
    print('Usage: copyFileWoComments.py sourceFileName destFileName')
    sys.exit(-1)

#print(sys.argv[0]) # prints copyArgFileWoComments.py
infilename = sys.argv[1]
outfilename = sys.argv[2]

try:
    inFile = open(infilename, 'r')
    outFile = open(outfilename, 'w')
except IOError:
    print('Error performing file IO')
    traceback.print_exc()
    #sys.exit(-1)

print('completed')
```

Handling Exceptions

```
C:\Users\Rohit\Lec29Scripts>python3 handleError.py missingFile.txt missingFileCopy.txt
```

Error performing file IO

Traceback (most recent call last):

```
  File "C:\Users\Rohit\Lec29Scripts\handleError.py", line 16, in <module>
    inFile = open(inFilename, 'r')
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'missingFile.txt'
```

completed

Handling Exceptions

```
$ if python -c "import sys; sys.exit(-1)"; then echo "Everything fine"; else echo "Not good"; fi
```

Not good

```
$ if python -c "import sys; sys.exit(0)"; then echo "Everything fine"; else echo "Not good"; fi
```

Everything fine

<https://stackoverflow.com/questions/44893807/i-want-to-know-what-exactly-sys-exit-1-returns-in-python>

Program Efficiency

Program Efficiency

- During design, we need to choose data structure and algorithms
- Implement in various languages
 - python, java, C, C++, Perl, ...
- Run on different hardwares and operating systems
 - 1 GHz, 2 GHz, 2.8 GHz
- Range of data sizes
 - Students, Population in Country, Globe
- How to decide which data structure or algorithm is best?

Program Efficiency

- How to decide which data structure or algorithm is best?
- Time Complexity: Decide on basis of “basic” operations
 - comparison, arithmetic, copy, etc. of scalar variables
 - count each as 1 unit
- E.g. find average of N Numbers

	Work
- sum = 0	# 1 unit
- For i in 1 to N do	# $2*N$ units, $i += 1, i \leq N$
- sum = sum + A[i]	# $1*N$ units
- avg = sum/N	# 1 unit
- Total Work	# $2 + 3N$ units

Time Complexity

Eg 1: Find average of N numbers Work

```
sum = 0                                          1  
for I in 1 to N do                            2*N  
    sum = sum + A[I]                            1*N  
avg = sum / N                                    1
```

Total Work = $2+3N$

Simplify problem = Abstraction

Time Complexity

- Eg 2: Find average of numbers read so far

```
• def GetAvg(A, K): # avg of A[1..K]
    sum = 0
    for I in 1 to K do
        sum = sum + A[I]
    return sum / K
```

Total Work = $2+3K$

Time Complexity

- Eg 2: Find average of numbers read so far

```
• def GetAvg(A, K): # avg of A[1..K]
    sum = 0
    for I in 1 to K do
        sum = sum + A[I]
    return sum / K
```

1
2*K
1*K
1

Total Work = $2+3K$

```
for I in 1 to N do
    avg[I] = GetAvg(A, I)
```

Time Complexity

- Eg 2: Find average of numbers read so far

```
def GetAvg(A, K): # avg of A[1..K]
    sum = 0
    for I in 1 to K do
        sum = sum + A[I]
    return sum / K
```

1
2*K
1*K
1

Total Work = $2+3K$

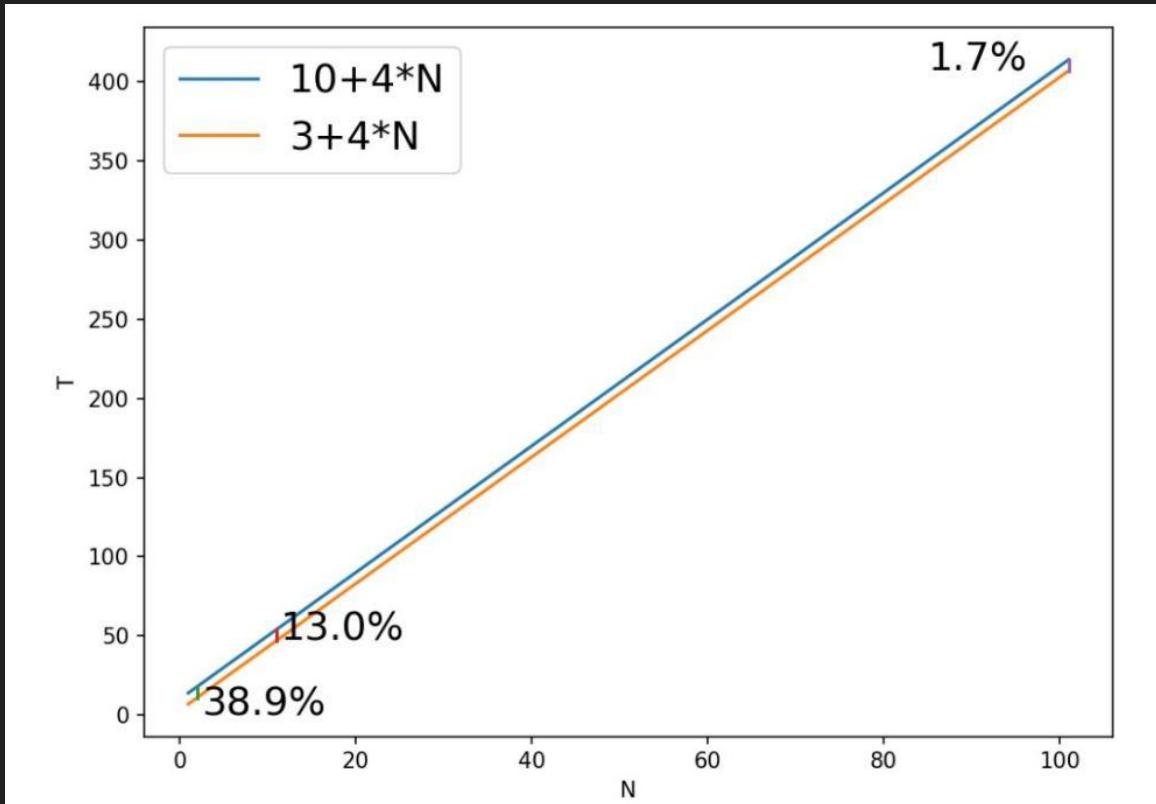
```
for I in 1 to N do
    avg[I] = GetAvg(A, I)
```

$$\begin{aligned}\text{Total Work} &= (4+3)+(4+3*2)+(4+3*3)+\dots+(4+3*N) \\ &= 5.5N + 1.5N^2\end{aligned}$$

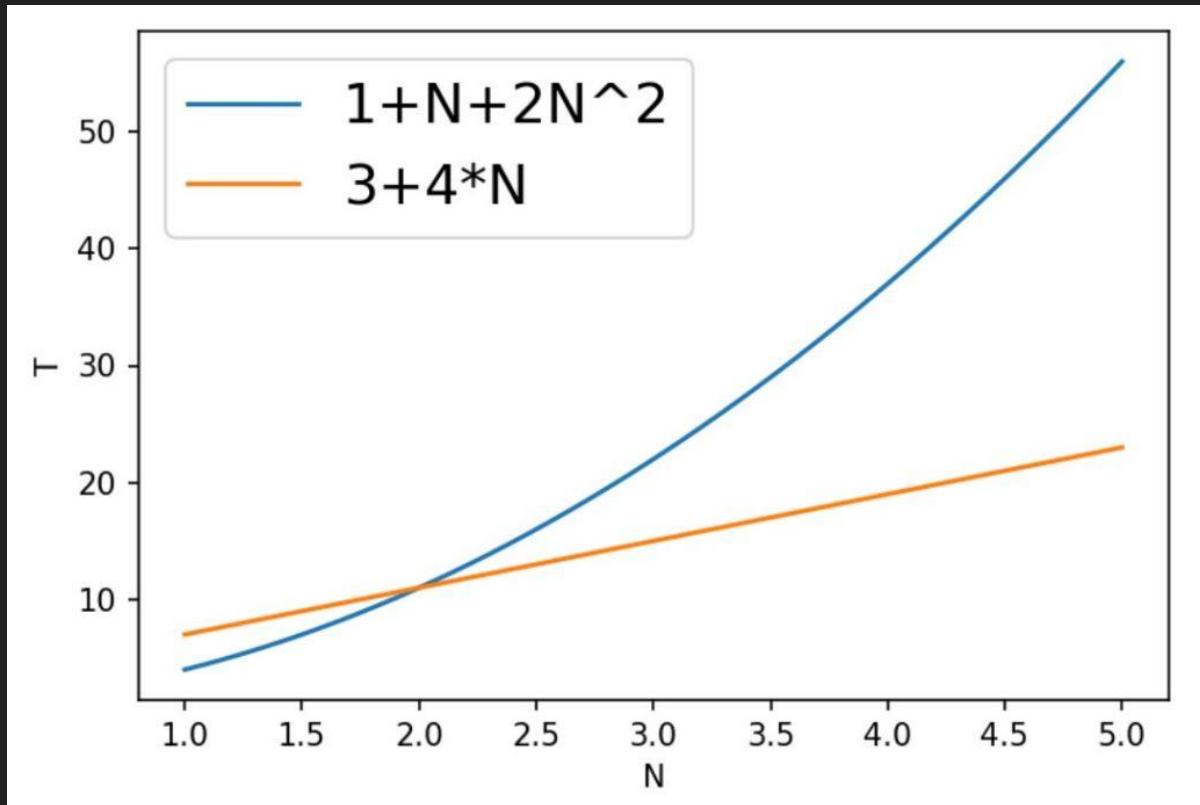
Time Complexity

- Following $T(N)$ have ~ same efficiency for large N
- $T_1(N) = 2 + 4N$, $T_2(N) = 10 + 4N$
- $T_1(N) = 1 + N + 2N^2$, $T_2(N) = 10 + 20N + 2N^2$
- For following $T_2(N)$ is greater than $T_1(N)$ for large N
- $T_1(N) = 10 + 4N$, $T_2(N) = 1 + N + 2N^2$
- $T_1(N) = 200 + 400N$, $T_2(N) = 1 + N + 2N^2$

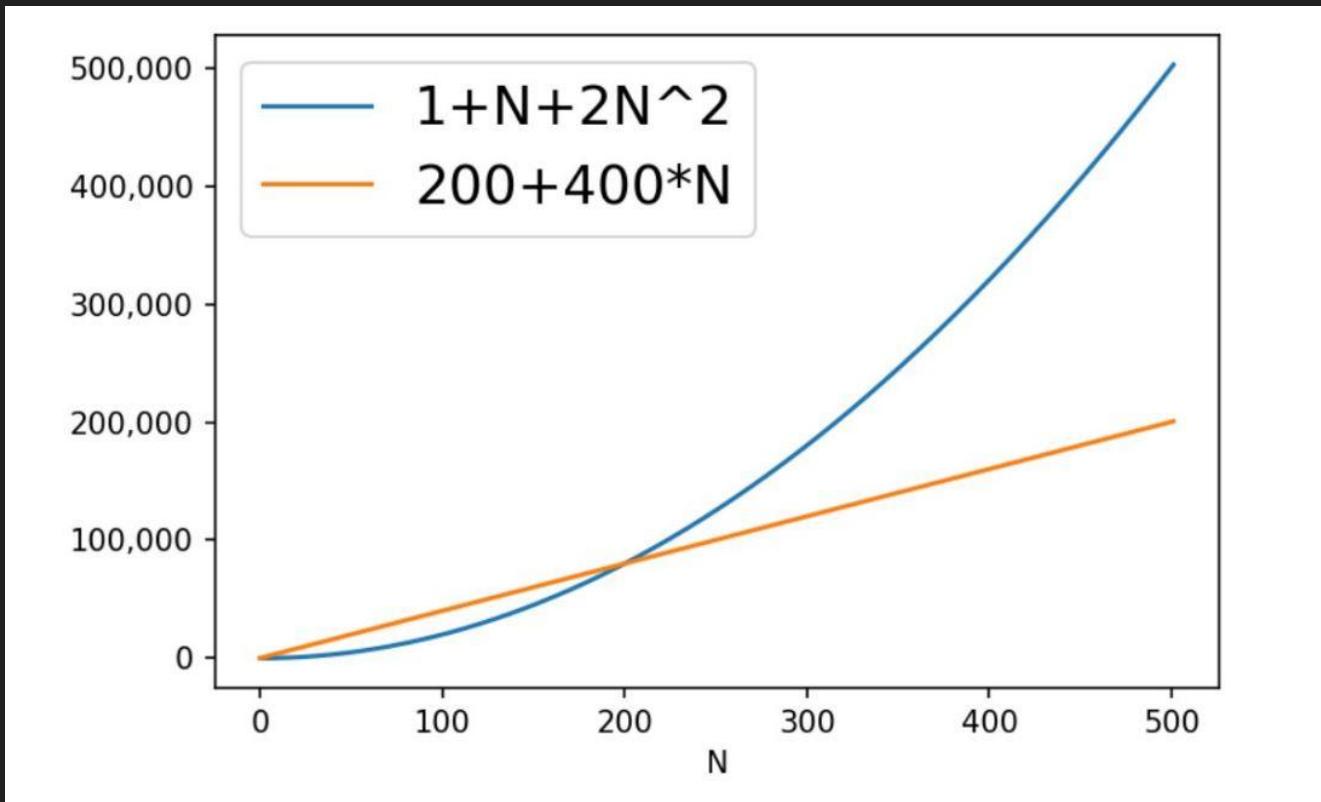
Time Complexity



Time Complexity



Time Complexity



Time Complexity

- In general
- For following $T_2(N) > T_1(N)$ for $N \geq N_0$ (some value)
- $T_1(N) = a + bN$
- $T_2(N) = c + dN + eN^2$
- a, b, \dots, e are independent of N
- Asymptotic Complexity = $O()$: Considers Highest power of N
- If T_1 is $O(N)$, and T_2 is $O(N^2)$, then T_1 is faster than T_2 .

Time Complexity

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Object Oriented Programming

```
class Bike:

    def handle(self):
        print("Used for acceleration.")

    def break(self):
        print("Used to stop bike.")
```

Object Oriented Programming

- OOP is a programming paradigm
- that structures program so that
- properties and behaviour
- are bundled together, into objects.
- Another paradigm:
- procedural programming
 - Divide a program into functions
 - (this is mostly what we have been using so far.)

```
class Bike:  
  
    def handle(self):  
        print("Used for acceleration.")  
  
    def break(self):  
        print("Used to stop bike.")
```

Representing Data

- Simple data types like integers, strings
- More complex data: eg. employee information – Name, Empld, DoB, Location, Dept, DoJ, reportingTo
- How to represent this type of data?
- We can use a collection of lists, like this:
- E1 = ['Rajat Kumar', '1023456', '03-10-1980', 'Sales', '4556234']
- E2 = ['Meena Singh', '4556234', '14-05-1971', 'Sales', '76654556']
- and so on
- Problems with this include:
 - Keeping track of indices
 - Hard to maintain
 - This is not the use of a list

Rajat Kumar
1023345
09-10-1980
Delhi
Sales
12-3-2019
3445235

Meena Singh
3445235
09-10-1971
Mumbai
Sales
12-3-2001
5662234

Object Oriented Programming: Classes

- Classes are used to create user-defined data structures
- A class is a template for creating an instance, called an object
- Classes usually have
 - attributes, and
 - methods

```
class Bike:  
  
    def handle(self):  
        print("Used for acceleration.")  
  
    def break(self):  
        print("Used to stop bike.")
```

Object Oriented Programming: Classes

```
class Car:
```

Object Oriented Programming: Classes

```
class Car:  
  
    # Class attributes:  
    manufacturer = 'Maruti'
```

Object Oriented Programming: Classes

```
class Car:

    # Class attributes:
    manufacturer = 'Maruti'

    # Constructor method
    def __init__(self, name, year):
        """Initializes a car object with name and year attributes"""

        self.name = name
        self.year = year
```

Object Oriented Programming: Classes

```
class Car:

    # Class attributes:
    manufacturer = 'Maruti'

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year
```

Object Oriented Programming: Classes

```
class Car:

    # Class attributes:
    manufacturer = 'Maruti'

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

t = Car('Wagon R', 2019)
print(t.name)
print(t.year)
```

Object Oriented Programming: Classes

```
class Car:

    # Class attributes:
    manufacturer = 'Maruti'

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

t = Car('Wagon R', 2019)
print(t.name)
print(t.year)

r = Car('Alto', 2010)
print(r.name)
print(r.year)
```

Object Oriented Programming: Classes

```
class Car:  
  
    # Class attributes:  
    manufacturer = 'Maruti'  
  
    # Constructor method  
    def __init__(self, name, year):  
        # Instance attributes:  
        self.name = name  
        self.year = year  
  
t = Car('Wagon R', 2019)  
print(t.name)  
print(t.year)  
  
r = Car('Alto', 2010)  
print(t.name)  
print(t.year)  
print(r.manufacturer)  
print(t.manufacturer)
```

Wagon R
2019
Wagon R
2019
Maruti
Maruti

Object Oriented Programming: Classes

```
class Car:  
  
    # Class attributes:  
    manufacturer = 'Maruti'  
  
    # Constructor method  
    def __init__(self, name, year):  
        # Instance attributes:  
        self.name = name  
        self.year = year  
  
t = Car('Wagon R', 2019)  
print(t.name)  
print(t.year)  
  
r = Car('Alto', 2010)  
print(t.name)  
print(t.year)  
print(r.manufacturer)  
print(t.manufacturer)
```

```
# attributes can be changed:  
t.manufacturer = 'Honda'  
t.name = 'City'  
print(t.manufacturer, t.name, t.year)
```

Honda City 2019

Incorrect Use of Class Variable

```
class Car:

    # mistaken use of class variable
    dimensions = [] # L, W, H

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)

t = Car('Wagon R', 2019)

r = Car('Alto', 2010)

t.add_dimensions(3655) # Length 3655 mm Width 1620 mm Height 1675 mm
r.add_dimensions(3445) # 3445 mm L x 1490 mm W x 1475 mm H
print(t.dimensions)
```

Incorrect Use of Class Variable

```
class Car:  
  
    # mistaken use of class variable  
    dimensions = [] # L, W, H
```

Incorrect Use of Class Variable

```
class Car:

    # mistaken use of class variable
    dimensions = [] # L, W, H

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year
```

Incorrect Use of Class Variable

```
class Car:

    # mistaken use of class variable
    dimensions = [] # L, W, H

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)
```

Incorrect Use of Class Variable

```
class Car:

    # mistaken use of class variable
    dimensions = [] # L, W, H

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)

t = Car('Wagon R', 2019)

r = Car('Alto', 2010)
```

Incorrect Use of Class Variable

```
class Car:

    # mistaken use of class variable
    dimensions = [] # L, W, H

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)

t = Car('Wagon R', 2019)

r = Car('Alto', 2010)

t.add_dimensions(3655) # Length 3655 mm Width 1620 mm Height 1675 mm
r.add_dimensions(3445) # 3445 mm L x 1490 mm W x 1475 mm H
print(t.dimensions)
```

[3655, 3445]

Correct Use of Attributes

```
class Car:

    # Constructor method
    def __init__(self, name, year):
        # Instance attributes:
        self.name = name
        self.year = year
        # correct use via attributes
        self.dimensions = [] # L, W, H

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)

t = Car('Wagon R', 2019)

r = Car('Alto', 2010)

t.add_dimensions(3655) # Length 3655 mm Width 1620 mm Height 1675 mm
r.add_dimensions(3445) # 3445 mm L x 1490 mm W x 1475 mm H
print(t.dimensions)
```

[3655]

Instance methods can be used, for example, to check for validity of input

```
class Car:

    # Constructor method
    def __init__(self, name, year):
        # Instance attribute
        self.name = name
        self.year = year
        # correct use via attributes
        self.dimensions = [] # L, W, H

    # Instance method
    def add_dimensions(self, dim):
        self.dimensions.append(dim)

    # Dunder method
    def __str__(self):
        return f"{self.name} has dimensions {self.dimensions}"

t = Car('Wagon R', 2019)

t.add_dimensions(3655) # Length 3655 mm Width 1620 mm Height 1675 mm
t.add_dimensions(1620)
t.add_dimensions(1620)
print(t)
```

Listing Object Attributes and Methods

built-in `dir()` function

Inheritance

Parent class



Child class 1

Child class 2

(Usually) more specialized

Child classes can override or extend attributes from the parent class



Hair colour from mother
But you can override it

Add an attribute that your parents
don't have (eg. learning a new
language)

Inheritance

```
class Car:  
  
    # Constructor method  
    def __init__(self, owner, year):  
        # Instance attributes:  
        self.owner = owner # owner's name  
        self.year = year
```

Inheritance

```
class Car:

    # Constructor method
    def __init__(self, owner, year):
        # Instance attributes:
        self.owner = owner # owner's name
        self.year = year

    # Instance method
    def color(self, color = 'white'):
        return f"{self.owner}'s car has color {color}"

# child class mercedesbenz, parent class in argument
class mercedesbenz(Car):
    pass
```

Inheritance

```
class Car:

    # Constructor method
    def __init__(self, owner, year):
        # Instance attributes:
        self.owner = owner # owner's name
        self.year = year

    # Instance method
    def color(self, color = 'white'):
        return f"{self.owner}'s car has color {color}"

# child class mercedesbenz, parent class in argument
class mercedesbenz(Car):
    pass

c1 = Car('Ajaya', 2022)
c2 = mercedesbenz('Kishori', 2023)
print(c1.color())
print(c2.color())
```

Ajaya's car has color white
Kishori's car has color white

Inheritance

```
class Car:

    # Constructor method
    def __init__(self, owner, year):
        # Instance attributes:
        self.owner = owner # owner
        self.year = year

    # Instance method
    def color(self, color = 'white'):
        return f"{self.owner}'s car has color {color}"

# child class mercedesbenz, parent class in argument
class mercedesbenz(Car):
    # overriding the base class's method
    def color(self, color = 'gray'):
        return f"{self.owner}'s car has color {color}"

c1 = Car('Ajaya', 2022)
c2 = mercedesbenz('Kishori', 2023)
print(c1.color())
print(c2.color())
```

Ajaya's car has color white
Kishori's car has color gray

Inheritance

```
class Car:  
  
    # Constructor method  
    def __init__(self, owner, year):  
        # Instance attributes:  
        self.owner = owner # owner  
        self.year = year  
  
    # Instance method  
    def color(self, color = 'white'):  
        return f"{self.owner}'s car has color {color}"  
  
# child class mercedes  
class mercedesbenz(Car):  
    # overriding the base class's color method  
    def color(self, color = 'gray'):  
        return f"{self.owner}'s car has color {color}"  
  
c1 = Car('Ajaya', 2022)  
c2 = mercedesbenz('Kishori', 2023)  
print(c1.color())  
print(c2.color())
```

Ajaya's car has color white
Kishori's car has color gray

print(isinstance(c1, Car)) # True
print(isinstance(c2, Car)) # True
print(isinstance(c1, mercedesbenz)) # False
print(isinstance(c2, mercedesbenz)) # True

Inheritance

```
class Car:  
  
    # Constructor method  
    def __init__(self, owner, year):  
        # Instance attributes:  
        self.owner = owner # owner  
        self.year = year  
  
    # Instance method  
    def color(self, color = 'white'):  
        return f"{self.owner}'s car has color {color}"  
  
# child class mercedes  
class mercedesbenz(Car):  
    # overriding the base class's color method  
    def color(self, color = 'gray'):  
        return f"{self.owner}'s car has color {color}"  
  
c1 = Car('Ajaya', 2022)  
c2 = mercedesbenz('Kishori', 2023)  
  
print(c1.color())  
print(c2.color())
```

Ajaya's car has color white
Kishori's car has color gray

Using Empty Class to Bundle Few Names Attributes

```
class student:  
    pass  
  
Sam = student() # create empty student record  
print(dir(student))  
  
Ram = student() # create empty student record  
  
# fill the fields of records  
student.name = 'Ram Sharma'  
student.dept = 'computer science'  
student.styfund = 20000  
  
print(Ram.name)  
print(Sam.name)  
print(dir(student))
```

Using Empty Class to Bundle Few Names Attributes

```
class student:  
    pass
```

```
Sam = student() # create empty student record  
print(dir(student))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
 '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
 '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__']
```

Ram Sharma

Ram Sharma

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
 '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
 '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__', 'dept', 'name', 'styfund']
```

Stacks and Queues

- Stacks and queues are dynamic data structures
- Delete operation is pre-specified
- Stack: last-in,first-out (LIFO)
- Queue: first-in first-out (FIFO)

Stack Last In First Out (LIFO)

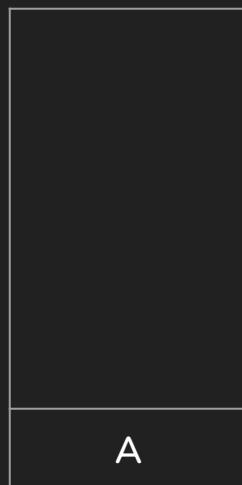
Empty Stack

(Stack is created)



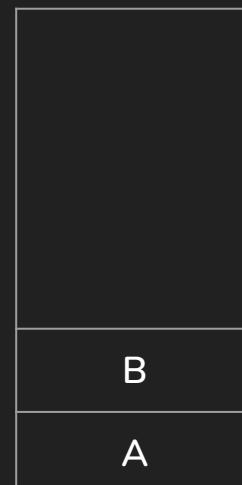
push A

top



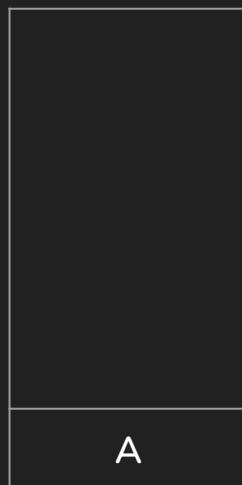
push B

top

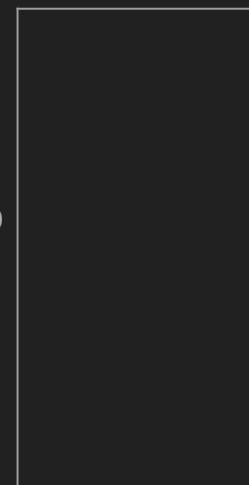


pop

top



pop



Stack Last In First Out (LIFO)

Desirables:

```
>>> s1 = Stack()  
>>> s1.push(1) # push 1 to stack's top  
>>> s1.push(2) # push 2 to stack's top  
>>> s1.push(3) # push 3 to stack's top  
>>> print(s1) # print stack, * indicates top
```

```
[1, 2, 3*]
```

```
>>> s1.pop()
```

```
3
```

```
>>> print(s2) # print stack, * indicates top  
[1, 2*]
```

Stack Last In First Out (LIFO)

```
class Stack():

    # create empty stack
    def __init__(self):
        #
        self.contents = []
        self.element = None

    def __str__(self):
        foo = str(self.contents)
        foo = foo[0:-1] + '*'
        return foo

    def push(self, x):
        self.contents.append(x)

    def pop(self):
        self.element = self.contents.pop()
        return self.element

    def peek(self):
        self.element = self.contents[-1]
        return self.element
```

```
s = Stack()
s.push('Monday')
s.push('Tuesday')
s.push('Wednesday')
print(s)
ele = s.pop()
print(ele, s)
ele = s.peek()
print(ele, s)
```

```
['Monday', 'Tuesday', 'Wednesday'*]
Wednesday ['Monday', 'Tuesday'*]
Tuesday ['Monday', 'Tuesday'*]
```

Stack Last In First Out (LIFO)

- Based on the application, a stack can have overflow and underflow.
- Overflow: Pushing onto a stack that is already full
- Underflow: Popping from an empty stack

Think: Python lists grow as needed. How will you implement overflow in the stack defined in the previous slide?

HW: Implement isEmpty(), len(), isOnTop(x) for the Stack class.

Stack Applications

- Undo in text editors
- Uses in various other algorithms
- Function calls in programs

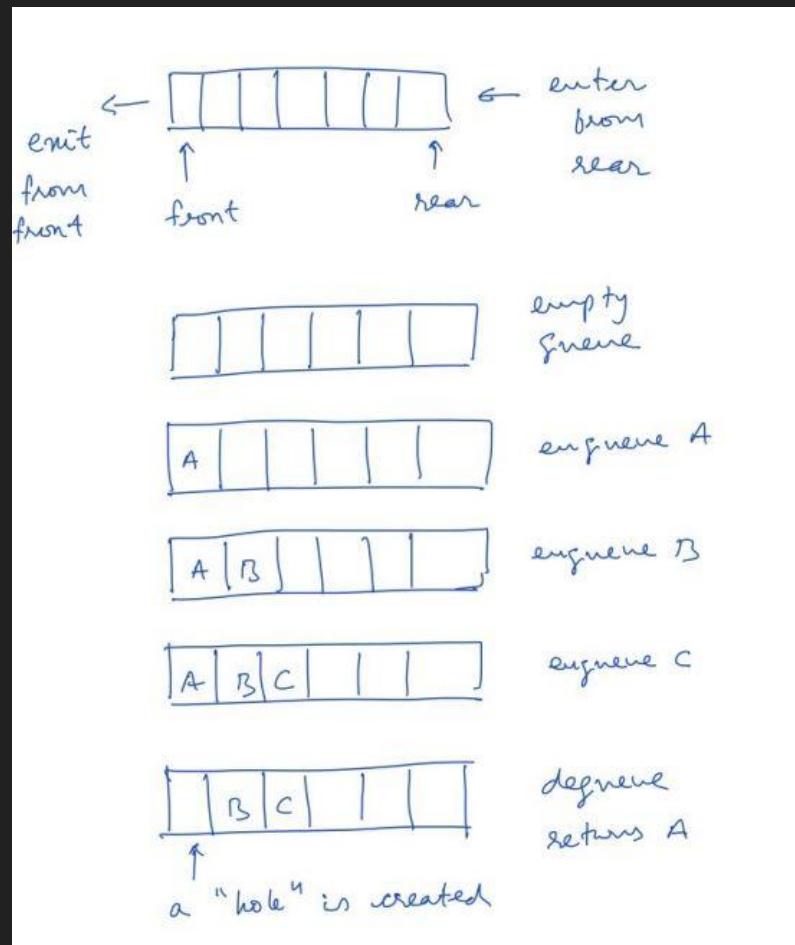
```
in h 3  
in f 5  
in main 3
```

Where to go after executing `h()`? Need to come back to line in `g()`. Use a stack to keep track.

```
def f():  
    x = 5  
    g()  
    print('in f', x)  
  
def g():  
    x = 7  
    h()  
  
def h():  
    print('in h', x)  
  
x = 3  
f()  
print('in main', x)
```

Queues (FIFO)

- FIFO: First In First Out
- Using a standard list is inefficient for queues
- Removal from front requires movement of data to fill up the space created



Queues in Python

- Python provides `collections.deque`
- Generalization of stacks and queues
- Provides many methods, in addition to basic stack and queue operations
- `deque` can also be used as a stack
- See more here:
<https://docs.python.org/3/library/collections.html#collections.deque>

```
from collections import deque

queue = deque()
queue.append("Terry")
queue.append("Graham")
queue.popleft()
queue.popleft()
```

Queues in Python

- Python provides `collections.deque`
- Generalization of stacks and queues
- Provides many methods, in addition to basic stack and queue operations
- `deque` can also be used as a stack
- See more here:
<https://docs.python.org/3/library/collections.html#collections.deque>

```
from collections import deque

queue = deque()
queue.append("Terry")
queue.append("Graham")
queue.popleft()
queue.popleft()
```

Using `deque` as a queue (FIFO)

Queues Applications

- Job scheduling in operating systems
- As auxiliary data structures in various algorithms

References

<https://www.techopedia.com/definition/6597/computing>

<https://slideplayer.com/slide/5334907/>

Lot of material is build on top of last year IC152 slides by Prof. Padmanabhan Rajan

<https://pythonnumericalmethods.berkeley.edu/notebooks/chapter09.02-Floating-Point-Numbers.html>

Python Programming for the Absolute Beginner, Third Edition Paperback – by Michael Dawson

Many Probability Slides are prepared using class notes by Prof. Ajit Rajwade (IIT Bombay)

Thank You