

# Python Fundamentals - Part 1

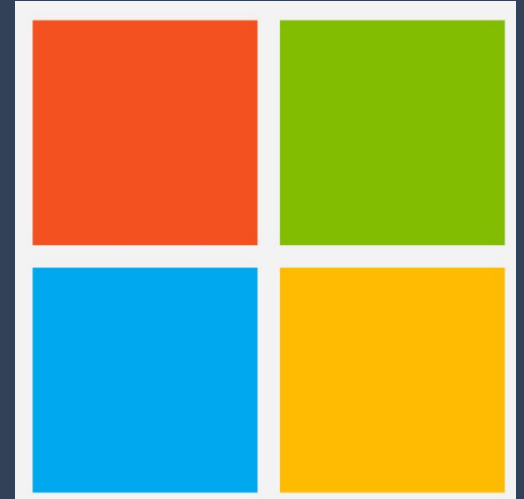
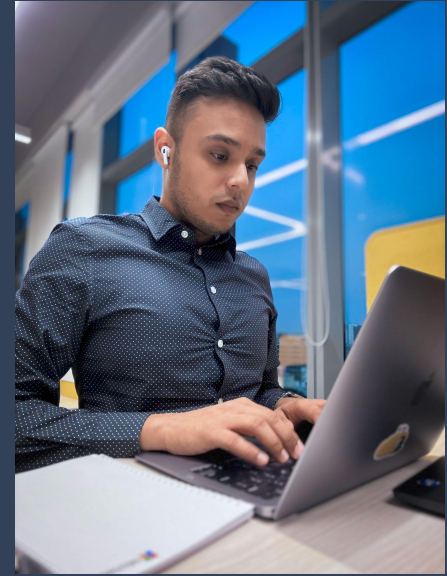


**Instructor: Anshaj Khare**

[Anshaj Khare | LinkedIn](#)

# Introduction – Anshaj Khare

- ✓ I have been working on data science since 2017
- ✓ I've worked for 4 startups before joining Microsoft
- ✓ I have been working at Microsoft for about 2.5 years.
- ✓ In my career, I've worked as a backend engineer, a devops engineer, a data engineer and as a Machine learning engineer

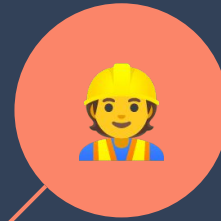


**Welcome to today's class, before we begin,  
pop into the chat:**

Your Name



Role



Location

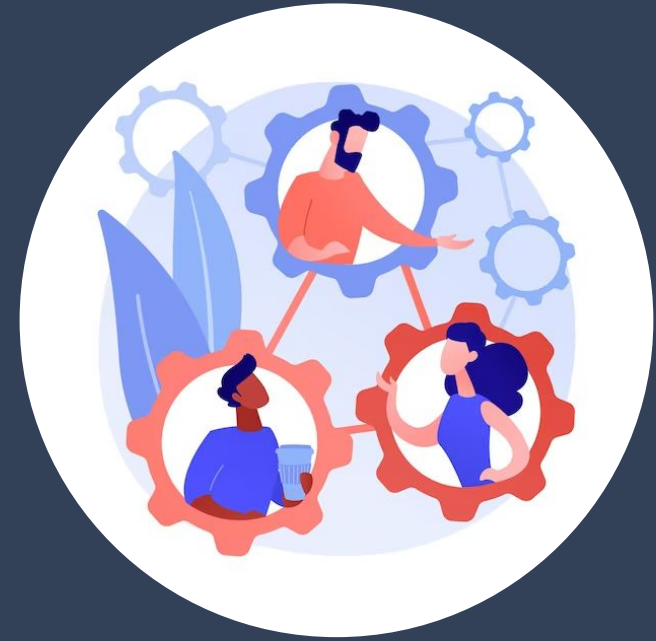


Company



# Optimize Your Experience

- ✓ Interact with your instructors via Sunday live class.
- ✓ Don't be shy to speak up and get clarifications !



- ✓ Solve your assignments, MCQs and other assessments and get feedback (Thursday review session)
- ✓ Use your resources! It's your experience – what you put in is what you'll get out.

# Today's Agenda



Break

Break

Break

1



Intro to  
Python,  
Data Types  
& Strings

2



Conditionals  
,Functions,  
OOPS in  
Python

3



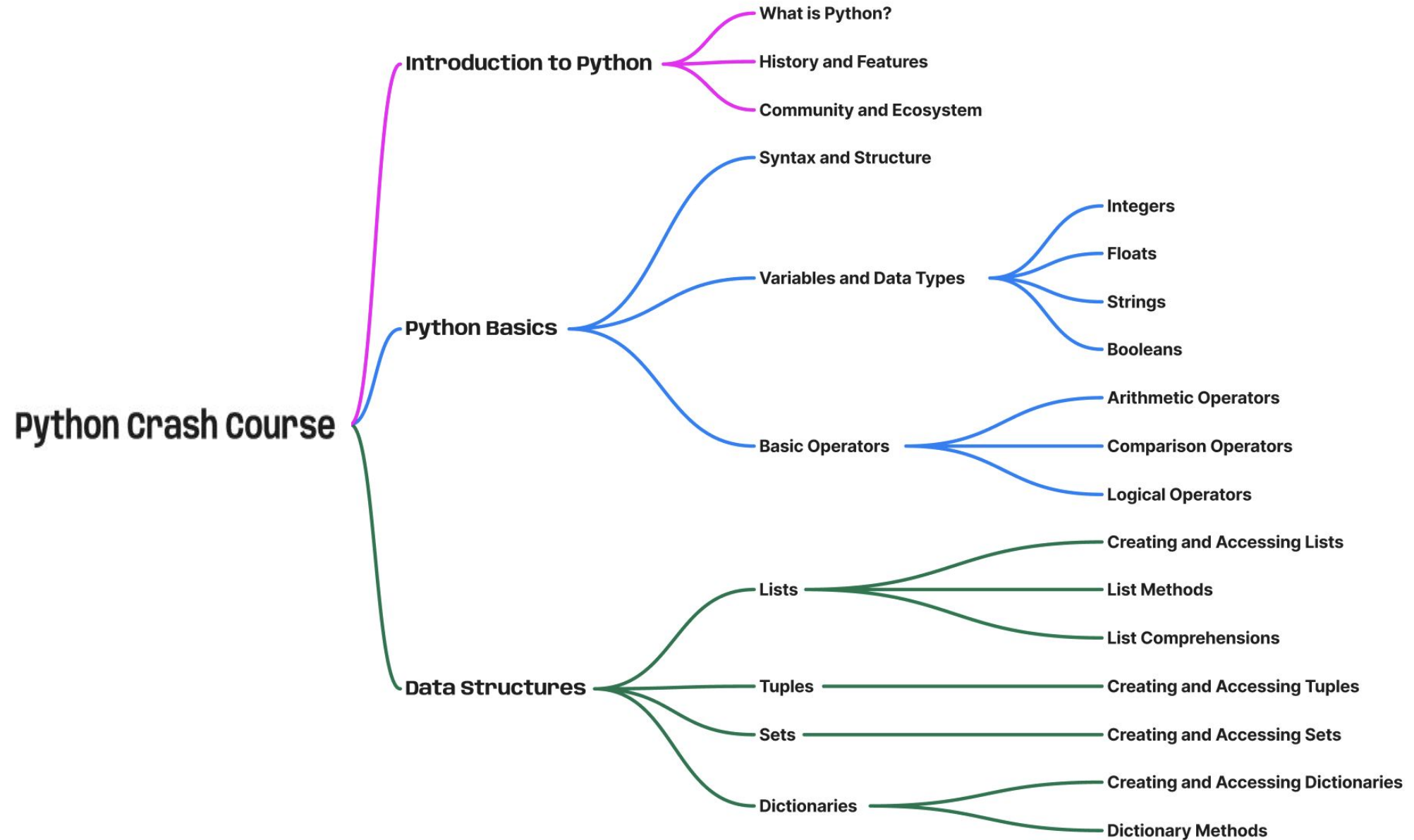
NumPy

4

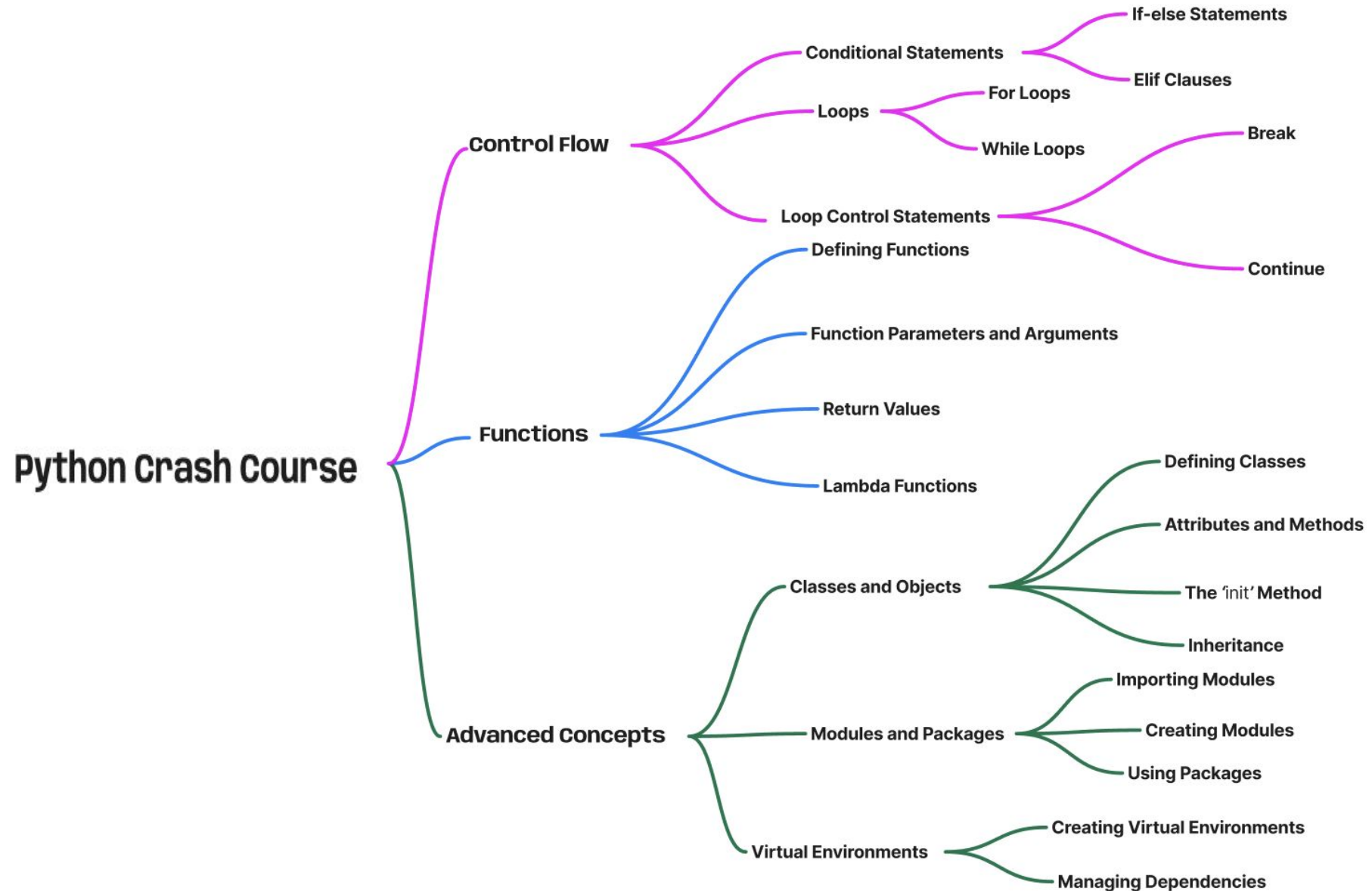


Pandas,  
Dataframe  
Operations

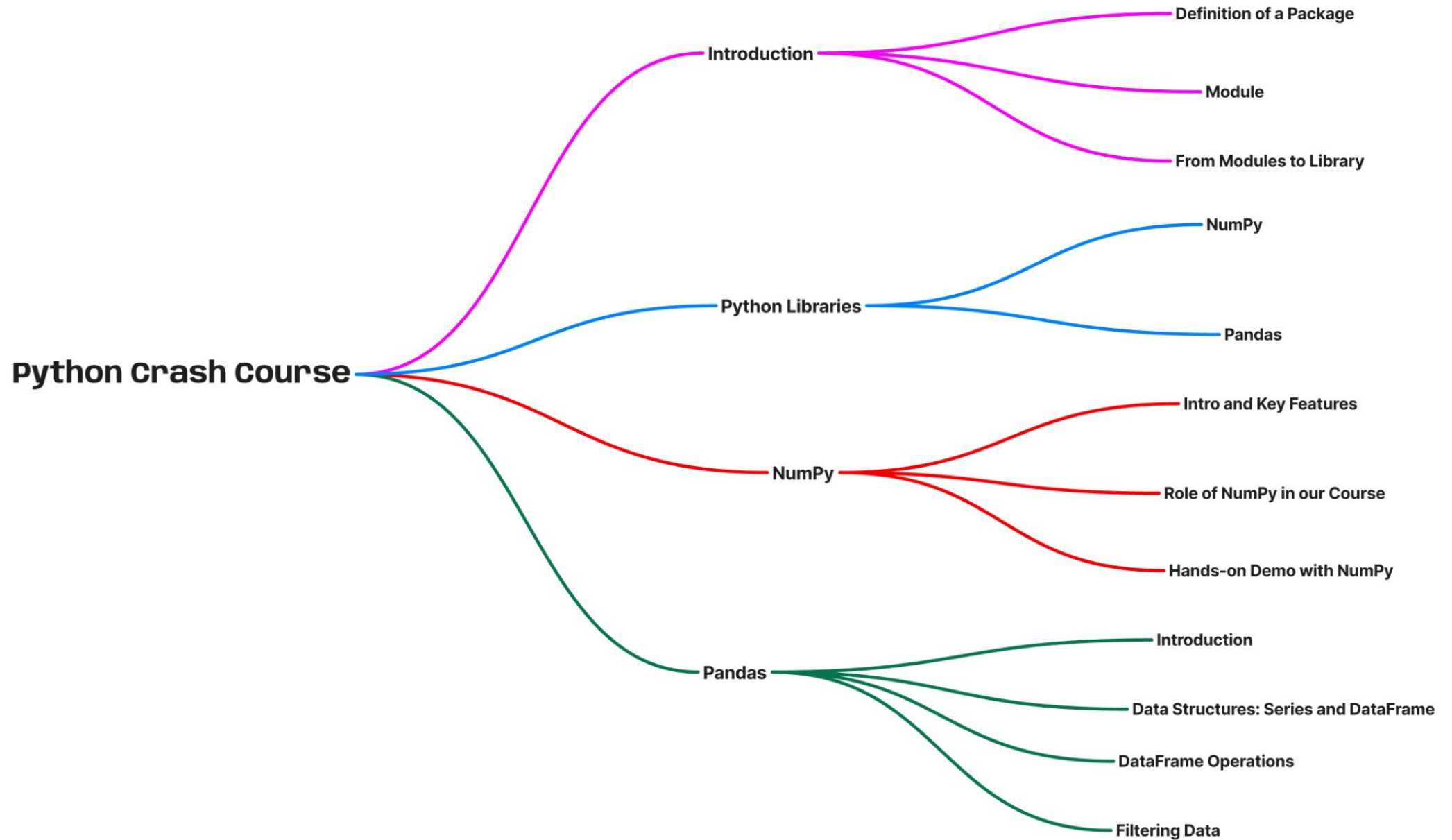
# Session Mindmap - Part 1



# Session Mindmap - Part 2



# Session Mindmap - Part 3





# How is Today's Content Relevant to My Job Role?

- **PMs:** Learn to identify Python-based AI features for product development.
- **TPMs:** Understand Python's role in implementing AI features and managing projects.
- **SDEs:** Develop Python skills for integrating AI capabilities into applications
- **Engineering Managers:** Discover how to leverage Python for integrating AI in projects.
- **DevOps Engineers:** Understand Python's infrastructure requirements for deploying AI models.

# Frequently Asked Question - 1

**I don't have to code in my job role, why am I learning Python here? How will it help me?**

Don't worry, you don't have to learn Python to the extent of a Python Developer. You only need to be able to comprehend the code that would be used in the upcoming modules in the course.

In GenAI, Python is just a tool! With this Crash Course, you would understand just enough on how to operate the tool and that's all you need!

# Frequently Asked Question - 2

**I have never coded in Python before. Will I become an expert in Python after completing this module?**

This is a Python Crash Course - tailored specifically to equip you with enough knowledge to help you navigate through the Applied GenAI Course smoothly.

Generative AI technologies are implemented through Python, and you will encounter multiple coding demos in Python throughout the course. This Python Crash Course includes just enough details so that the coding demos in the further modules are comprehensible.

If you have never coded in Python before, you may need to refer to external resources too. If you need help with the resources, please contact [studentsupport@interviewkickstart.com](mailto:studentsupport@interviewkickstart.com)

**If you are already familiar with Python programming, this module may seem like a basic refresher for you - and that's exactly what it's meant to be!**

# Today's Agenda



Break

Break

Break

1



Intro to  
Python,  
Data Types  
& Strings

2



Conditionals  
,Functions,  
OOPS in  
Python

3



NumPy

4



Pandas,  
Dataframe  
Operations

# Introduction to Python

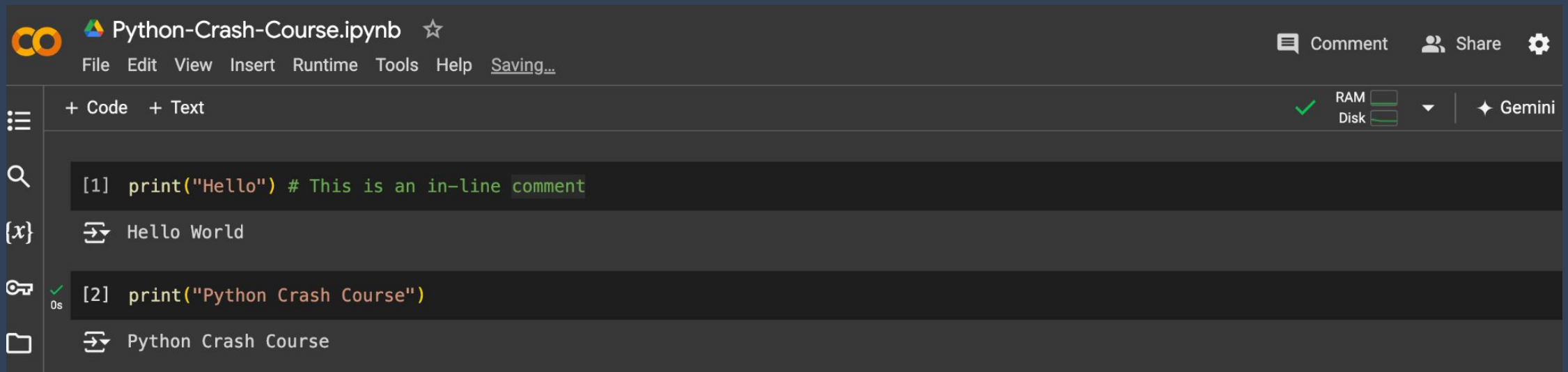
# Introduction

- Python is a versatile and powerful programming language used in various domains. From web development to data science, Python has become a language of choice for many.
- Python is a **high-level**, **interpreted**, and object-oriented programming language.
- Python was created by Guido Van Rossum and released in 1991.
- There are several Integrated Development Environments (**IDEs**) available for coding in Python, including PyCharm, Visual Studio Code, Spyder, and **Jupyter**



# Why Google Colab?

- It allows you to write your code in separate blocks or cells, and you can execute these cells one at a time.



# Role of Python in Generative AI

- **Primary Language for AI:** Python's simplicity and readability make it the go-to language for AI development, supported by libraries like TensorFlow, PyTorch, and Keras.
- **Generative Models:** Python is used to implement various generative models such as GANs, VAEs, and Transformers for applications like image, text, and audio generation.
- **Integration and Deployment:** Seamless integration with APIs, web services, and cloud platforms (e.g., AWS, Google Cloud) for scalable deployment of AI models.
- **Extensive Libraries:** Essential libraries like NumPy and Pandas facilitate data manipulation and preprocessing tasks critical for AI development.
- **Vibrant Community:** A large, active community and abundant educational resources support ongoing learning and development in Python for generative AI.



# Variables and Data Types

# A Problem-Based Learning Approach

ChatGPT 4 ▾



what is generative AI?



Generative AI refers to artificial intelligence technologies that can generate new content, ranging from text, images, and music to code and more. These systems learn from large datasets to produce outputs that are often indistinguishable from those created by humans. Some popular examples include:

1. **Text Generation:** AI models like GPT (Generative Pre-trained Transformer) can compose emails, write articles, generate creative fiction, or code.
2. **Image Generation:** Models such as DALL-E and Stable Diffusion create images from textual descriptions, enabling users to generate highly detailed and specific images based on prompts.
3. **Music and Audio:** AI systems can compose music or generate synthetic voices.
4. **Code Generation:** Tools like GitHub Copilot assist in writing code by suggesting entire lines or blocks of code.

These technologies are based on neural networks, particularly deep learning models like transformers, which have shown remarkable effectiveness in understanding and generating complex patterns. The potential applications of generative AI are vast, ranging from creative arts to technical fields like software development and data analysis.



# A Problem-Based Learning Approach

We provide a question to the LLM (Large Language Model).



LLM (Generative AI)



The model generates a response based on the given question.

# A Problem-Based Learning Approach



```
● ● ●  
  
# Define the prompt for the AI model, asking it to summarize the provided text  
prompt = """  
Summarize the following text into a single sentence:  
Python is a versatile and powerful programming language used in various domains. From web development  
to data science, Python has become a language of choice for many.  
Python is a high-level, interpreted, and object-oriented programming language.  
Python was created by Guido Van Rossum and released in 1991.  
Python's readability and simplicity make it an ideal language for beginners.  
It's widely used in industry, making it a valuable skill for future opportunities.  
"""  
  
# Generate the summary response by calling LLM  
response = call_LLM(prompt)  
  
# Print the generated summary  
print(response)
```

# Basic Python Programming - Variables and Data Types

- **Variables:**

- Store values for later use
- No need to declare data type in advance

- **Data types:**

- Common types: int, float, str, bool, list, tuple, set and dictionary
- Automatically assigned based on the value

- **Integer:** Example: `a = 10`

- **Float:** Example: `b = 6.2`

**String:** Example: `c = "Matt"`

**Boolean:** Example: `d = True`

- **List:**

- An ordered, mutable **collection of items** enclosed in **square brackets []**.  
Items can be of different data types.
- Example: `c = [1, 2, 3, 4, 5]`

- **Tuple:**

- An ordered, immutable **collection of items** enclosed in **parentheses ()**. Items can be of different data types.
- Example: `d = (1, 2, 3, 4, 5)`

- **Set:**

- An unordered **collection of unique items** enclosed in curly braces **{}**.

Example: `c = {1, 2, 3, 4, 5}`

- **Dictionary:**

- A **collection of key-value pairs** enclosed in curly **braces** **{}**, where each key is unique and immutable, and values can be of any data type and are accessed using their keys

- Example: `d = {"name": "Alice", "age": 30}`

# Operators





```
x = 10
y = 3
# Arithmetic operators
print("x + y =", x + y)
print("x - y =", x - y)
# Comparison operators
print("x == y?", x == y)
print("x < y?", x < y)
```



```
X = 10
```

```
y = 20
```

```
name = "Alice"
```

```
# Logical operators
```

```
print("x > 0 and y > 0:", x > 0 and y > 0)
```

```
print("not x < 0:", not x < 0)
```

```
# Other operators
```

```
print("'A' in name:", 'A' in name)
```

```
print("x is y:", x is y)
```

# Lists

# Python List - Overview and Basic Operations

- **Definition:**

- Ordered, mutable collection of items
- Can contain elements of different data types
- Created using square brackets []

- **Usage:**

- Create variable-length and mutable sequences of objects
- Store objects of any type
- Store multiple types of objects together



# Characteristics of a Python List

The various characteristics of a list are -

- **Ordered:** Lists maintain the order in which the data is inserted.

```
fruits = ["apple", "banana", "cherry"]  
print(fruits) # Output: ['apple', 'banana', 'cherry']
```

- Accessing Items using their index (Zero-based, Negative indexing).

```
first_item = fruits[1]  
print(first_item) # Output: 'banana'
```

- **Dynamic:** List can expand or shrink automatically to accommodate the items accordingly.
- **Duplicate Elements:** Lists allow us to store duplicate data

- **Heterogenous:** Lists can store elements of various data types.

```
mixed_list = ["apple", 42, 3.14, [1, 2, 3]]  
print(mixed_list) # Output: ['apple', 42, 3.14, [1, 2, 3]]
```

- **Mutable:** In list element(s) are changeable. It means that we can modify the items stored within the list.

```
# Adding an item  
fruits.append("date")  
print(fruits) # Output: ['apple', 'banana', 'cherry', 'date']
```

```
# Removing an item  
fruits.remove("banana")  
print(fruits) # Output: ['apple', 'cherry', 'date']
```

```
# Changing an item  
fruits[1] = "blueberry"  
print(fruits) # Output: ['apple', 'blueberry', 'date']
```

# Slicing in Python

Slicing is a technique in Python used to extract a subset of elements from a sequence (such as a list, tuple, or string) by specifying a range of indices.

`sequence[start:stop:step]`

start: The starting index of the slice (inclusive). Defaults to 0 if not specified.

stop: The ending index of the slice (exclusive). The slice includes elements up to, but not including, this index.

step: The step size, or the interval between indices in the slice. Defaults to 1 if not specified.

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# Slicing from index 2 to 5
```

```
subset = numbers[2:6]  
print(subset) # Output: [2, 3, 4, 5]
```

```
# Slicing with step size of 2
```

```
subset = numbers[1:8:2]  
print(subset) # Output: [1, 3, 5, 7]
```

```
# Define a string
```

```
text = "Hello, World!"
```

```
# Slicing from index 7 to the end
```

```
substring = text[7:]  
print(substring) # Output: "World!"
```

```
# Slicing from the beginning to index 5
```

```
substring = text[:5]  
print(substring) # Output: "Hello"
```

**Do you wonder how negative indexing works in Python Slicing?**





# Python Slicing - Negative Indexing

- Negative indices:
  - Can be used for slicing to refer to elements from the end of the sequence



```
# Create a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using negative index to slice the list from the second-
to-last element to the fifth-to-last element (exclusive)
slice_negative = numbers[-5:-2]
print("Negative index slice:", slice_negative) # Output:
[6, 7, 8]
```

# Tuples

# Python Tuple - Overview & Characteristics

The various characteristics of a Tuple are -

- **Ordered Collection**

```
fruits = ("apple", "banana", "cherry")  
print(fruits) # Output: ('apple', 'banana', 'cherry')
```

- Accessing Items using their index (zero-based, negative).

```
first_item = fruits[0]  
print(first_item) # Output: 'apple'
```

- **Immutable Collection**: We can not add, remove or change an item in Tuples.

- **Different Data Types**

```
mixed_tuple = ("apple", 42, 3.14, (1, 2, 3))  
print(mixed_tuple) # Output: ('apple', 42, 3.14, (1, 2, 3))
```

# Set

# Characteristics of a Python Set

The various characteristics of a set are -

- **Unordered Collection:**

```
fruits_1 = {"apple", "banana", "cherry"}  
fruits_2 = {"apple", "cherry", "banana"}
```

- **Unique Items::**

```
fruits = {"apple", "banana", "cherry", "apple"}  
print(fruits) # Output: {'apple', 'banana', 'cherry'} (duplicates are removed)
```

- **Mutable Collection:**

```
# Adding an item  
fruits.add("date")  
print(fruits) # Output: {'apple', 'banana', 'cherry', 'date'}
```

```
# Removing an item  
fruits.remove("banana")  
print(fruits) # Output: {'apple', 'cherry', 'date'}
```

# Characteristics of a Python Set

Different Data Types (Sets cannot contain mutable elements like lists or dictionaries)

```
mixed_set = {"apple", 42, 3.14, (1, 2, 3)}  
print(mixed_set)  # Output: {42, 3.14, 'apple', (1, 2, 3)}
```

# Dictionary in Python

# Characteristics of Dictionaries in Python

## Key-Value Pairs

```
student_ages = {"Alice": 25, "Bob": 22, "Charlie": 23}  
print(student_ages) # Output: {'Alice': 25, 'Bob': 22, 'Charlie': 23}
```

## Unique

- If a dictionary contains duplicate keys, the last key-value pair will be used.

```
student_ages = {"Alice": 25, "Bob": 22, "Charlie": 23, "Alice": 26}  
print(student_ages) # Output: {'Alice': 26, 'Bob': 22, 'Charlie': 23}
```

## Accessing Values::

```
age_of_alice = student_ages["Alice"]  
print(age_of_alice) # Output: 26
```



# Characteristics of Dictionaries in Python

## Adding and Modifying Items

```
# Adding a new key-value pair
student_ages["David"] = 24
print(student_ages) # Output: {'Alice': 26, 'Bob': 22, 'Charlie': 23, 'David': 24}
```

```
# Modifying an existing key-value pair
student_ages["Alice"] = 27
print(student_ages) # Output: {'Alice': 27, 'Bob': 22, 'Charlie': 23, 'David': 24}
```

## Different Data Types

- Keys must be immutable (e.g., strings, tuples).
- Values can be of any data type.

```
mixed_dict = {"name": "Alice", "age": 25, "is_student": True, "grades": [85, 90, 92]}
print(mixed_dict)
```

# Practical Examples

1. **Storing Data with Different Attributes:** Dictionaries are ideal for storing data with different attributes, such as information about users, products, or any entities with multiple properties.

```
user = {  
    'name': 'John Doe',  
    'age': 30,  
    'email': 'john@example.com',  
    'city': 'New York'  
}
```

1. **Retrieving Information Efficiently:** Dictionaries offer fast lookups, allowing you to retrieve information quickly using keys.

```
# Accessing user's email  
email = user['email']
```

# Practical Examples

3. **Mapping Unique IDs to Objects:** Dictionaries are useful for mapping unique identifiers to objects, facilitating efficient retrieval of objects based on their IDs.

```
# Mapping student IDs to their information
students = {
    1001: {'name': 'Alice', 'age': 20},
    1002: {'name': 'Bob', 'age': 22},
    # More students...
}
```

3. **Configuration Settings:** Dictionaries are commonly used to store configuration settings for applications.

```
# Application settings
settings = {
    'debug': True,
    'log_level': 'INFO',
    'timeout': 30
}
```

# Strings

# Python String - Defining and Using in Python

- Ordered, immutable sequence of characters
- Enclosed in single or double quotes (" or """)



```
# Using single quotes for a simple string
single_quoted_string = 'Hello, World!'

# Using double quotes for a simple string
double_quoted_string = "Hello, World!"

# Using double quotes to avoid escape character
double_without_escape = "It's a beautiful day."
single_with_escape = 'It\'s a beautiful day.'

# Multi-line string with triple quotes
multi_line_string = """
This is a multi-line string.
It can span multiple lines without needing any escape characters.
"""
```

# Important String Methods



*Converts all characters in the string to lowercase.*

`str.lower():`

Example: `"Hello".lower()` returns `"hello"`.

*Converts all characters in the string to uppercase.*

`str.upper():`

Example: `"hello".upper()` returns `"HELLO"`.

*Capitalizes the first character of the string and lowercases the rest.*

`str.capitalize():`

Example: `"hello world".capitalize()` returns `"Hello world"`.

*Converts the first character of each word to uppercase and the rest to lowercase.*

`str.title():`

Example: `"hello world".title()` returns `"Hello World"`.

*Removes any leading and trailing whitespace*

`str.strip():`

Example: `" hello ".strip()` returns `"hello"`.

*Removes any leading whitespace*

`str.lstrip():`

Removes **any** leading whitespace.

Example: `" hello ".lstrip()` returns `"hello "`.



# Important String Methods



*Removes any trailing whitespace*

`str.rstrip():`

Example: `" hello ".rstrip()` returns `"hello"`.

*Splits the string into a list of substrings based on a specified delimiter*

`str.split():`

Example: `"hello world".split()` returns `["hello", "world"]`.

*Joins a list of strings into a single string with a specified delimiter*

`str.join():`

Example: `" ".join(["hello", "world"])` returns `"hello world"`.



# Advanced String Operations in Python

- String Concatenation:

- ```
# String Concatenation into one string.  
str1 = "Hello"  
str3 = "World"  
concatenated_string = str1 + " " + str3  
print(concatenated_string) # Output: Hello World
```

- String Formatting

- ```
name = "Alice"  
age = 30  
formatted_string = f"My name is {name} and I am {age} years old."  
print(formatted_string) # Output: My name is Alice and I am 30 years old.
```

- ```
formatted_string = "My name is {} and I am {} years old.".format(name, age)  
print(formatted_string) # Output: My name is Alice and I am 30 years old.
```



# Advanced String Operations in Python



```
text = """
Python is a versatile and powerful programming language used in various domains. From web development to
data science, Python has become a language of choice for many.
Python is a high-level, interpreted, and object-oriented programming language.
Python was created by Guido Van Rossum and released in 1991.
Python's readability and simplicity make it an ideal language for beginners.
It's widely used in industry, making it a valuable skill for future opportunities.
"""

# Define the prompt for the AI model, asking it to summarize the provided text
prompt = f"""
Summarize the text delimited by triple backticks into a single sentence.
```{text}```
"""

# Generate the summary response by calling LLM
response = call_LLM(prompt)

# Print the generated summary
print(response)
```



# Quiz

How do you access the second element of a list named `my_list`?

- A** `my_list(1)`
- B** `my_list[1]`
- C** `my_list.second()`
- D** `my_list[2]`

# Quiz - Solution

How do you access the second element of a list named `my_list`?

- A** `my_list(1)`
- B** `my_list[1]`
- C** `my_list.second()`
- D** `my_list[2]`

# Quiz

What is the output of the following code snippet?

```
my_dict = {'a': 1, 'b': 2, 'c': 3}  
print(my_dict['d'])
```

**A** 0

**B** None

**C** IndexError: 'd'

**D** KeyError: 'd'

# Quiz - Solution

What is the output of the following code snippet?

```
my_dict = {'a': 1, 'b': 2, 'c': 3}  
print(my_dict['d'])
```

**A**

0

**B**

None

**C**

IndexError: 'd'

**D**

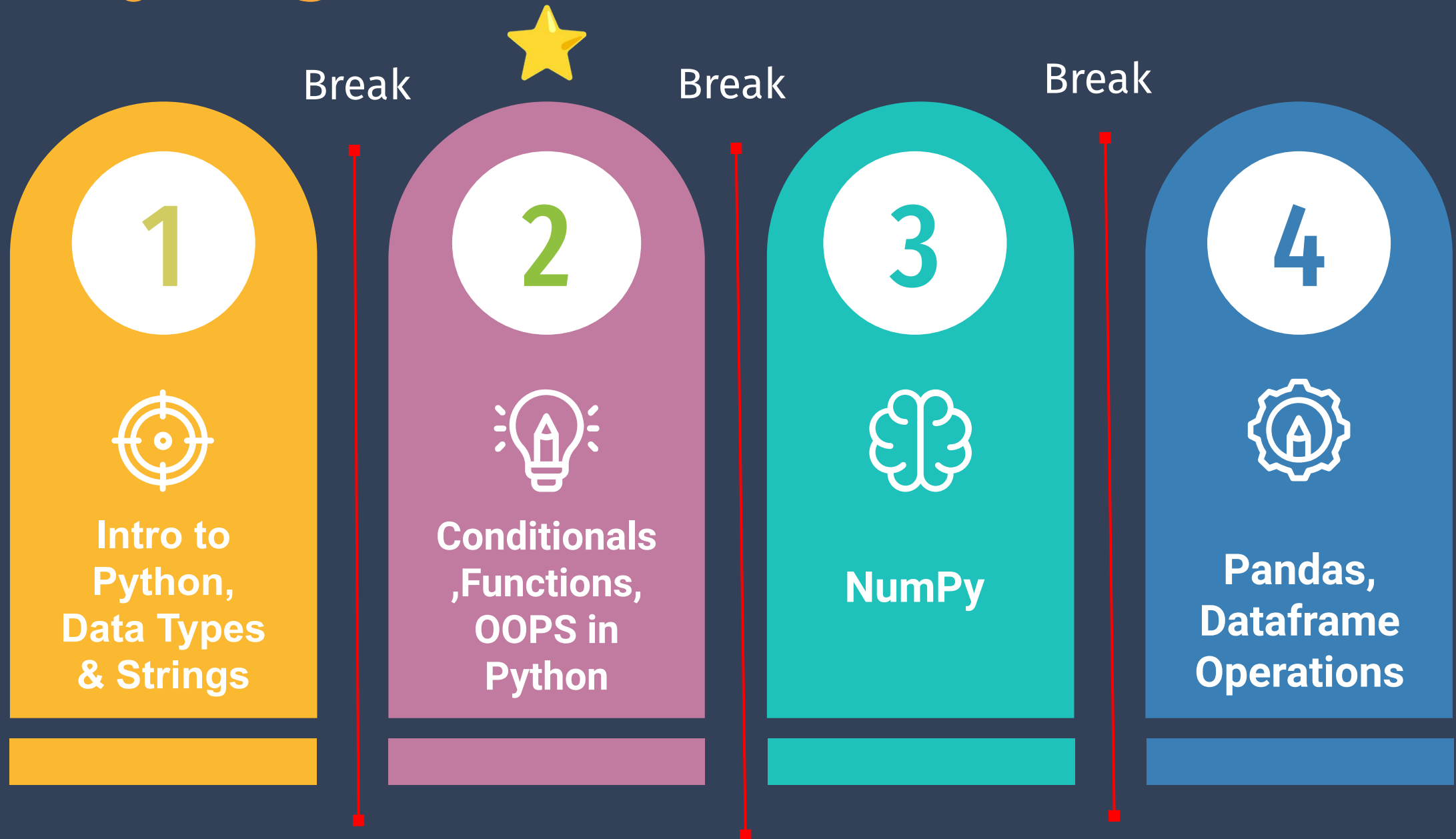
KeyError: 'd'

Break

Time!



# Today's Agenda





# Conditionals in Python : If-else statements

# Indentation in Python

Indentation in Python is a fundamental aspect of the language's syntax. Unlike many other programming languages that use braces `{}` to delimit blocks of code, Python uses indentation to define the scope and structure of loops, conditionals, functions, and other control structures.

## Consistency is Crucial:

1. All lines of code in a block must be indented the same amount.
2. Python does not enforce a specific number of spaces, but the convention is to use 4 spaces per indentation level

## Indentation Levels:

2. Each indentation level indicates a new block of code.
3. Indentation helps to visually distinguish blocks of code that are nested inside other blocks.

## Indentation for Control Structures:

1. Indentation is used to define the scope of loops (for, while), conditionals (if, elif, else), functions (def), classes (class), and other constructs.

# Conditional Statements in Python

- ✓ The **if statement** checks a condition, and if the condition is True, it executes the block of code that follows.
- ✓ The **elif statement** checks another condition if the previous if or elif condition was False. It can be used multiple times.
- ✓ The **else statement** is executed if none of the preceding if or elif conditions are True.

```
x = 10
if x > 5:
    print("x is greater than 5")
```

---

```
x = 10
if x > 15:
    print("x is greater than 15")
elif x > 5:
    print("x is greater than 5 but less than
or equal to 15")
```

---

```
x = 3
if x > 5:
    print("x is greater than 5")
elif x > 2:
    print("x is greater than 2 but less
than or equal to 5")
else:
    print("x is 2 or less")
```

# Python If-Else Statements - Example



```
age = 18
```

```
if age >= 18:  
    print ("You are eligible to vote.")
```

```
else:  
    print ("You are not eligible to vote.")
```

# Loops in Python

# Python Loops - Types and Usage

In Python, loops are used to repeatedly execute a block of code as long as a certain condition is true or for a specified number of times.

1. **'for' Loops:** iterate over a sequence (such as a list, tuple, string, or range)
2. **'while' Loops:** used to repeatedly execute a block of code as long as a specified condition is true

# Python Loops - Types and Usage

Two types of loops:

- **For loop:** Iterate over a sequence (e.g., list, tuple, string)
- **While loop:** Execute code block while a condition is true

# Loops - For

## For Loop

Used for iterating over a sequence (such as a list, tuple, dictionary, set, or string).

## Syntax

```
for variable in sequence:  
    # code block to execute
```

## Example

```
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print(fruit)  
# Output:  
# apple  
# banana  
# cherry
```



# Loops - While

Repeatedly execute a block of code as long as a condition is true.

## Syntax

```
while condition:  
    # code block to execute
```

## Example

```
count = 1  
while count <= 3:  
    print(count)  
    count += 1
```

```
# Output:  
# 1  
# 2  
# 3
```

**Do you know what are loop  
control statements?**



# Python Loops - Loop Control

- **Loop control statements:**

- **break:** Exit the loop immediately
- **continue:** Skip the rest of the loop body and start the next iteration

## Loop Control : Using the break Statement



What will be the output of this code snippet?

```
• • •  
  
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

# Output of the previous code



# Output:

0  
1  
2

# Nested Loops in Python

- Nested loops:
  - Loops can be placed inside other loops to create complex looping structures



# Python Loops - Nested Loops



```
# List of items
items = ['a', 'b', 'c', 'd']
# List to hold the pairs
pairs = []
# Nested loop to create pairs
for i in items:
    for j in items:
        # Append the pair to the pairs list
        pairs.append((i, j))
# Print the list of pairs
print(pairs)
```

# Python Loops - Loop Control and Nested Loops

- Best practices:
  - Keep loop bodies simple and readable
  - Avoid excessive nesting





# Functions

# What are Functions?



??

# Python Functions - Definition and Syntax

## Functions:

- Reusable pieces of code that perform a specific task
- Improve code readability, modularity, and maintainability

## Function call:

- Invoke a function by its name, followed by parentheses containing arguments
- Defined using the def keyword.
- Parameters are variables listed inside the parentheses in the function definition.
- They allow you to pass information into functions.
- The return statement is used to exit a function and return a value.

```
def function_name(arguments):  
    # function body  
  
    return
```

Here,

- `def` - keyword used to declare a function
- `function_name` - any name given to the function
- `arguments` - any value passed to function
- `return` (optional) - returns value from a function

Let's see an example,

```
def greet():  
    print('Hello World!')
```

# Function

## Syntax

```
def function_name(parameters):  
    # code block to execute  
    return result
```

```
texts = [  
    "Python is a versatile ...  
]
```

```
summaries = []  
for text in texts:  
    prompt = f"""  
        Summarize the following text into a  
single sentence:  
        {text}  
        """  
    summary = call_LLM(prompt)  
    summaries.append(summary)  
print(summaries)
```

```
texts = [  
    "Python is a versatile ...  
]
```

```
def summarize_texts(texts):  
    summaries = []  
    for text in texts:  
        prompt = f"""  
            Summarize the following text into a single  
sentence:  
            {text}  
            """  
        summary = call_LLM(prompt)  
        summaries.append(summary)  
    return summaries
```

```
Summary_results = summarize_texts(texts)  
print(Summary_results )
```

**What's the difference between  
arguments and parameters?**



??

# Python Functions - Parameters and Arguments

- **Parameters vs. arguments:**
  - **Parameters:** Variables defined in the function's signature
  - **Arguments:** Values passed to the function when it's called



# Function Parameter and Argument

## ✓ Positional arguments:

Positional arguments are the most common type. They are passed to the function in the same order as the parameters are defined

```
def subtract(a, b):  
    return a - b
```

```
result = subtract(10, 5)  
print(result) # Output: 5
```

---

## ✓ Keyword arguments:

Keyword arguments are passed to the function by explicitly specifying the parameter name.

```
def greet(name, message):  
    print(f"{message}, {name}!")
```

```
greet(name="Alice", message="Hello")  
greet(message="Hi", name="Bob")
```

---

## ✓ Default parameters:

Default parameters allow you to specify default values for parameters. If an argument is not provided, the default value is used.

```
def greet(name, message="Hello"):  
    print(f"{message}, {name}!")
```

```
greet("Alice")  
greet("Bob", "Hi")
```

---



# Function Parameter and Argument

## ✓ Variable-length arguments:

### i. **\*args:**

Allows a function to accept any number of positional arguments and it collects all positional arguments into a tuple

```
def multiply(*args):  
    result = 1  
    for num in args:  
        result *= num  
    return result
```

```
print(multiply(1, 2, 3, 4))
```

```
# Output: 24
```

### i. **\*\*kwargs:**

Allows a function to accept any number of keyword arguments. It collects all keyword arguments into a dictionary

---

```
def print_info(**kwargs):  
    for key, value in kwargs.items():  
        print(f"{key}: {value}")
```

```
print_info(name="Alice", age=30,  
city="New York")
```

# **Python as an Object-Oriented Programming Language**

# What are Objects in the Programming World?

- In the real-world, objects are entities that exist - can be felt and touched. For example, a phone. It's an object that exists in the real world.
- In the programming world, an Object is an entity that exists in the memory

In Python, everything is an Object!

- The basic data types - Int, Float, String etc. are also objects and the data structures in Python - List, Set, Tuple, Dictionary are also objects.
- Every object is an instance of its class. We will learn more about classes in the later sections of this module.

# Attributes and Behavior of Objects

- All objects have certain properties or characteristics called as Attributes
- For example, a phone object in the real world may have attributes like color, size, width etc.
- In Python, the objects like list have attributes like length - that is its property or attributes that describes it

# Attributes and Behavior of Objects contd.

- Every object has some functions - it's the tasks that it can perform
- Just like a phone in a real world can perform functions like making calls, clicking pictures etc.
- In Python, the behavior of objects are called as **Methods**

# Class

A **class** is a **blueprint** for creating **objects**. You can think of an object derived from a class as a **container** that holds some variables (attributes) and functions (methods). These variables and functions might be related to each other.

```
class NameofClass:

    def __init__(self, var1, var2, ...):
        self.ovar1 = var1
        self.ovar2 = var2
        ....

    def func1(self, ...):
        ...

    def func2(self, ...):
        ...

    def func3(self, ...):

        a = self.ovar1 + self.ovar2
        b = self.func1(a)
        return b

obj1 = NameofClass(var1 = 12, var2 = 14, ...)
```

## class

### Attributes (Variables)

-var1

-var2

...

### Methods (Functions)

- \_\_init\_\_

- Func1

- Func2

- ...

# How Does This Help Me?

- As you work with Python for Generative AI, you would be working with different Python objects
- Understanding Attributes and Methods would help you work with the objects better

Break

Time!





# Today's Agenda

Break

Break

Break

1



Intro to  
Python,  
Data Types  
& Strings

2



Conditionals  
,Functions,  
OOPS in  
Python

3



NumPy

4



Pandas,  
Dataframe  
Operations



# Modules & Packages

# Module

- ✓ A module is a file containing Python code. It can define functions, classes, and variables. You can use it to organize your code into reusable pieces.
- ✓ Think of a module as a library in other languages like Java or C#.

```
# mymodule.py
```

```
def greet(name):  
    return f"Hello,  
{name}!"
```

```
def add(a, b):  
    return a + b
```

```
# main.py
```

```
import mymodule
```

```
print(mymodule.greet("Alice"  
) # Output: Hello, Alice!  
print(mymodule.add(3,  
4)) # Output: 7
```

```
python main.py
```

```
Hello, Alice!  
7
```

# Package

- ✓ A package is a **collection of modules** organized in directories that includes a special `__init__.py` file.
- ✓ Think of a package as a directory containing multiple related modules or libraries.
- ✓ `__init__.py` is a special Python file used to indicate that a directory is a Python package. It allows the package's modules to be imported together, providing initialization code for the package if needed.
- ✓ `pip` is the package installer for Python. It allows you to install and manage additional libraries and dependencies that are not included in the standard library.

```
mypackage/  
├── __init__.py  
├── module1.py  
└── module2.py
```

```
from mypackage import module1,  
module2  
  
print(module1.function1())  
print(module2.function2())
```

```
python main.py
```

```
Hello, Alice!  
7
```

# Virtual Environment

- ✓ A virtual environment is an isolated Python environment that allows you to manage dependencies for your projects independently of each other.
- ✓ To prevent dependency conflicts between projects and to keep the global Python environment clean.
- ✓ Steps: Create, activate, install dependencies, and verify.
- ✓ If you do not use a virtual environment, all the installed packages will be installed globally, potentially causing conflicts with other projects and cluttering your global Python environment.

```
python -m venv projectA_env  
python -m venv projectB_env
```

```
projectA_env\Scripts\activate
```

```
(projectA_env) pip install  
requests==2.24.0
```

# Library

- ✓ A library is a collection of **pre-compiled routines** that a program can use. These routines are often packaged into a format that can be easily included and used by various programs, facilitating code reuse and modular programming.
- ✓ Libraries can include various functions, classes, and variables that provide specific functionality. For example, a math library might include routines for mathematical operations, while a graphics library might include routines for rendering images.
- ✓ In Python, a library is typically a package that contains multiple modules. These modules can be related to one another and can be imported into your programs as needed.

```
mathlib/  
├── __init__.py  
├── addition.py  
└── subtraction.py
```

```
from .addition import add  
from .subtraction import subtract  
  
print(add(5, 3))  
print(subtract(5, 3))
```

```
import requests  
response =  
requests.get('https://api.example.  
com/data')  
  
print(response.json())
```

# Introduction to NumPy

# Role of NumPy in Applied Generative AI Course

- **Data Handling:** Efficient manipulation of large datasets with ndarray.
- **Mathematical Operations:** Essential for linear algebra and random number generation, crucial for AI model training.
- **Library Integration:** Seamless conversion to tensors in TensorFlow/PyTorch; foundation for Pandas, SciPy, and Matplotlib.
- **Community Support:** Extensive documentation and a vibrant, active community for continuous learning and development.





# NumPy

- NumPy stands for Numerical Python and is a foundational package for numerical computing in Python.
- This module will include only hands-on exercises on NumPy. We hope you got a chance to go through the video before attending the class.

# Time to Code NumPy



# Recap Quiz

## What distinguishes NumPy arrays from Python lists?

- A** NumPy arrays do not support operations like element-wise addition or multiplication, which Python lists do.
- B** Python lists are faster for numerical calculations than NumPy arrays.
- C** NumPy arrays can only store elements of the same data type, while Python lists can hold elements of different types.
- D** Python list is designed for efficiency on large arrays of data

# Recap Quiz - Solution

**What distinguishes NumPy arrays from Python lists?**

- A** NumPy arrays do not support operations like element-wise addition or multiplication, which Python lists do.
- B** Python lists are faster for numerical calculations than NumPy arrays.
- C** NumPy arrays can only store elements of the same data type, while Python lists can hold elements of different types.
- D** Python list is designed for efficiency on large arrays of data

# Today's Agenda

Break

1



Intro to  
Python,  
Data Types  
& Strings

Break

2



Conditionals  
,Functions,  
OOPS in  
Python

Break

3



NumPy



4

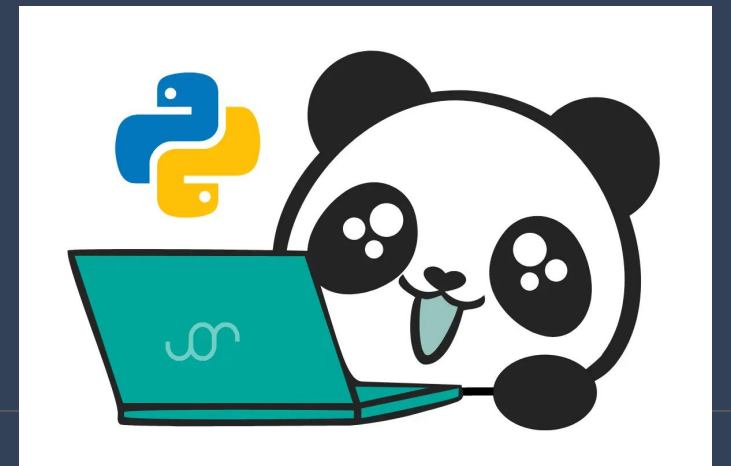


Pandas,  
Dataframe  
Operations

# Introduction to Pandas

# Introduction

- Pandas is a powerful Python library widely used for *data manipulation* and *analysis*.
- Pandas offers data structures and functions that simplify working with structured data, making it an essential tool in data science and analysis



# Why Pandas?

Using Pandas offers several benefits for data manipulation and data analysis:

- Efficient Data Handling
- Data Alignment
- Handling Missing Data
- Data Integration
- Flexible Data Transformation
- Integration with Other Libraries



# Data Structures in Pandas

- Pandas **Series** is a one-dimensional array with axis labels.
- Pandas **DataFrame** is two-dimensional data structure with labeled rows and columns.

Series A

|   | apples |
|---|--------|
| 0 | 3      |
| 1 | 2      |
| 2 | 0      |
| 3 | 1      |

+

Series B

|   | oranges |
|---|---------|
| 0 | 0       |
| 1 | 3       |
| 2 | 7       |
| 3 | 2       |

=

DataFrame

|   | apples | oranges |
|---|--------|---------|
| 0 | 3      | 0       |
| 1 | 2      | 3       |
| 2 | 0      | 7       |
| 3 | 1      | 2       |

# Filtering a DataFrame - Why do we need filtering?

- When we need to remove redundant or unnecessary data for some tasks
- When we want to filter customers based on certain criteria
- When we need to filter out rows or columns have missing values

# Filtering a DataFrame

- Filtering with ``loc`` and ``iloc`` methods
- Filtering by Selecting a Subset of Columns
- Filtering by condition(s)

# Filtering a DataFrame - `loc` and `iloc` methods

- **loc** uses row and column labels.
- **iloc** uses row and column indexes.

|           | column index |   | 0    | 1   | 2         |
|-----------|--------------|---|------|-----|-----------|
| row index | label        |   | Name | Age | Education |
| 0         | March        | 2 |      |     |           |
| 1         | April        | 3 |      |     |           |
| 2         | May          | 4 |      |     |           |
| 3         | June         | 5 |      |     |           |
| 4         | July         | 6 |      |     |           |

# Filtering a DataFrame - Selecting a Subset of Columns

- Select a single column
- Select multiple columns

|   | state | account_length | international_plan | total_intl_calls | churn |
|---|-------|----------------|--------------------|------------------|-------|
| 0 |       |                |                    |                  |       |
| 1 |       |                |                    |                  |       |
| 2 |       |                |                    |                  |       |

Select multiple columns

Select a column

# Filtering a DataFrame - Filtering by condition(s)

- Filtering by a condition
- Filtering by multiple conditions

The operators we can use in filtering by condition(s):

- ==: equal
- !=: not equal
- >: greater than
- >=: greater than or equal to
- <: less than
- <=: less than or equal to
- & : and
- | : or

# Handling Missing Values with Pandas

- Missing Value Types and Representation
- Finding the Missing Values
- Dropping Rows and Columns with Missing Values
- Replacing the Missing Values

# Handling Missing Values with Pandas

- Missing values are essentially data we don't have
- NaN : stands for “Not a Number”
  - Python's **None**
  - **np.nan**





## Finding the Missing Values

- `isna()` function: returns **True** to indicate a missing value
- `notna()` function: returns **False** for missing values

|   | state | account_length | churn |
|---|-------|----------------|-------|
| 0 | CA    | 314            | yes   |
| 1 | CO    | ?              | no    |
| 2 | ?     | 44             | yes   |

# Dropping Rows and Columns with Missing Values

- `dropna()` function:
  - ``axis``: 0 (default value, indicates row), 1
  - ``how``: “any” (default value), “all”
  - ``thresh``: a row or column that at least X non-missing values will be kept, otherwise will be dropped.

|   | state | account_length | churn |
|---|-------|----------------|-------|
| 0 | CA    | 314            | ?     |
| 1 | CO    | ?              | ?     |
| 2 | ?     | 44             | ?     |

# Replacing the Missing Values

- Dropping may not be the best option in many cases
- Replace missing values with:
  - the average value of the column
  - the most frequent value in the column
  - a random number
- `fillna()` function

|   | state | account_length | churn |
|---|-------|----------------|-------|
| 0 | CA    | 314            | Yes   |
| 1 | CO    | 100            | Yes   |
| 2 | CO    | 44             | no    |

# Sorting in a DataFrame

- Sort DataFrames based on their column features
- `sort_values()` : sort a DataFrame by one or more of its columns
  - First argument: either a column label or a list of column labels
  - ``ascending`` : ascending (default value), descending

|   | Name  | age |
|---|-------|-----|
| 0 | David | 33  |
| 1 | Emily | 20  |
| 2 | Emma  | 44  |



|   | Name  | age |
|---|-------|-----|
| 0 | Emily | 20  |
| 1 | David | 33  |
| 2 | Emma  | 44  |

# Data Manipulation with Pandas

- `drop()`: Remove rows or columns from a DataFrame
- `replace()`: Replace values within a DataFrame
- `apply()`: Applying a given function along an axis of a DataFrame or Series.
- Add a new column from current data

|   | user_id | account_length | total_orders | total_amt |
|---|---------|----------------|--------------|-----------|
| 0 | 1232as  | 31             | 4            | 156       |
| 1 | 1323wv  | 0              | 2            | 127       |
| 2 | 134pdf  | 0              | 3            | 135       |
| 3 | 342t6ff | 23             | 4            | 466       |

# Data Manipulation with Pandas

- `drop()`: Remove rows or columns from a DataFrame
- `replace()`: Replace values within a DataFrame
- `apply()`: Applying a given function along an axis of a DataFrame or Series.
- Add a new column from current data

|   | user_id | account_length | total_orders | total_amt |
|---|---------|----------------|--------------|-----------|
| 0 | 1232as  | 31             | 4            | 156       |
| 1 | 1323wv  | 0              | 2            | 127       |
| 2 | 134pdf  | 0              | 3            | 135       |
| 3 | 342t6ff | 23             | 4            | 466       |

# How Does This Help Me?

- Pandas will help you handle and analyze large datasets.
- Pandas provide data structures and operations for manipulating numerical tables and time series.



```
import pandas as pd

emotions.set_format(type="pandas")
df = emotions["train"][:]
df.head()
```

|   | text  | label |
|---|---|-------|
| 0 | i didnt feel humiliated                           | 0     |
| 1 | i can go from feeling so hopeless to so damned... | 0     |
| 2 | im grabbing a minute to post i feel greedy wrong  | 3     |
| 3 | i am ever feeling nostalgic about the fireplac... | 2     |
| 4 | i am feeling grouchy                              | 3     |

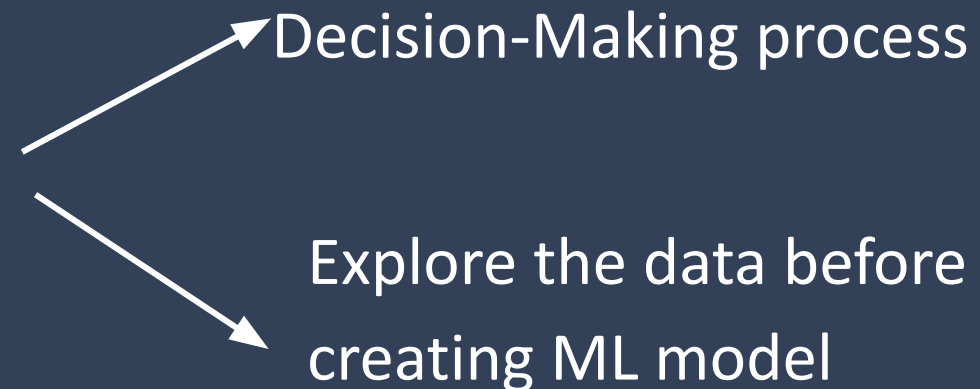
# Data Analysis with Pandas



# Data Analysis with Pandas

- Data Analysis: The process of inferring insights, discovering useful information, and drawing results from the data
  - `groupby()`: groups rows of a DataFrame based on one or more columns
  - `agg()`: aggregate data within groups created by ``groupby()``
  - `reset_index()`: reset the index of a DataFrame
  - `rename()`: change the labels (names) of rows or columns in a DataFrame
  - `idxmax()`: Return index of first occurrence of maximum over requested axis

|   | total_order | age |
|---|-------------|-----|
| 0 | 20          | 33  |
| 1 | 40          | 20  |
| 2 | 100         | 44  |



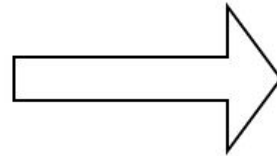
# Data Analysis with Pandas - groupby(), agg()

The `groupby` function creates the groups, but they don't provide any information unless we do some aggregations.

**DataFrame**

| product_group | product_code | price |
|---------------|--------------|-------|
| A             | 1001         | 9     |
| A             | 1002         | 14    |
| B             | 1101         | 21    |
| A             | 1003         | 12    |
| B             | 1104         | 19    |
| C             | 1201         | 7     |
| B             | 1105         | 25    |

**groupby("product\_group")**



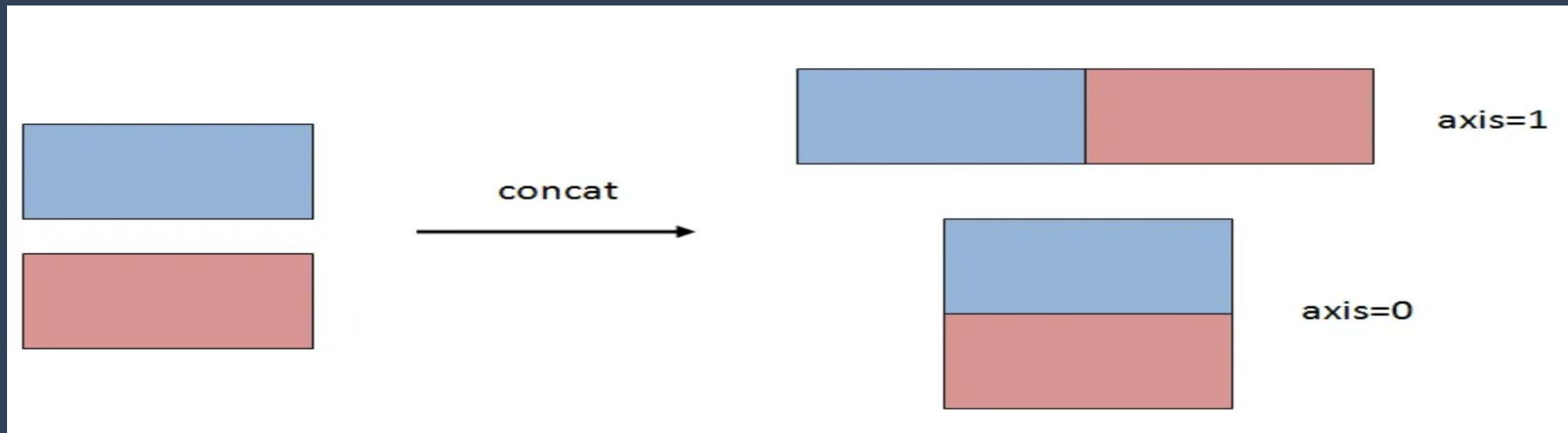
| product_group | product_code | price |
|---------------|--------------|-------|
| A             | 1001         | 9     |
| A             | 1002         | 14    |
| A             | 1003         | 12    |

| product_group | product_code | price |
|---------------|--------------|-------|
| B             | 1101         | 21    |
| B             | 1104         | 19    |
| B             | 1105         | 25    |

| product_group | product_code | price |
|---------------|--------------|-------|
| C             | 1201         | 7     |

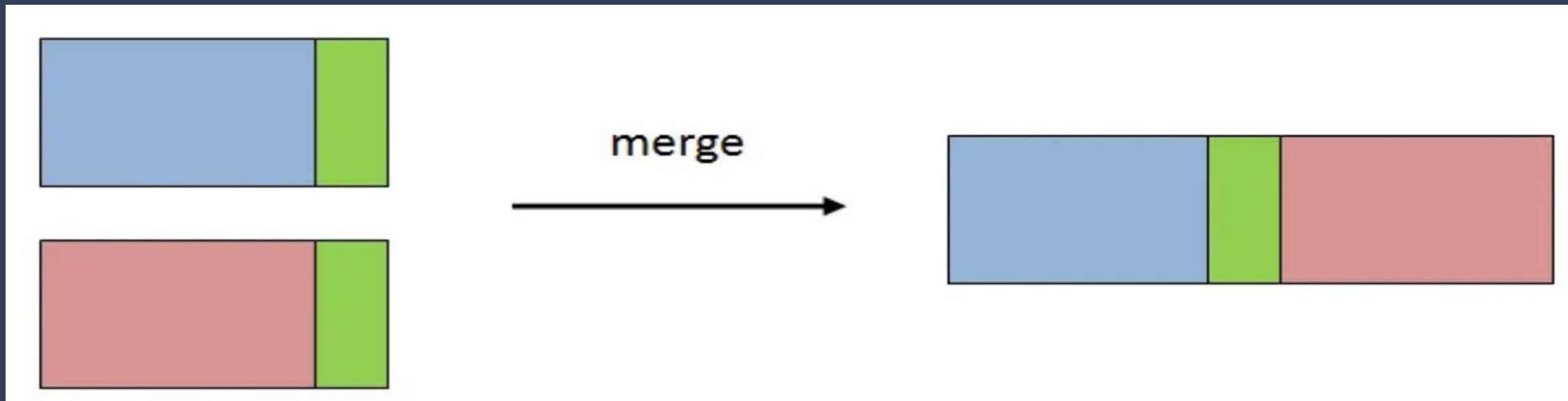
# Combining DataFrames with Pandas - The concat function

- Concatenation is the process of combining DataFrames along a particular axis (either vertically or horizontally).
- When you concatenate DataFrames, you're essentially stacking them on top of each other (vertically) or side by side (horizontally).
- The resulting DataFrame maintains the original column names and indexes of the individual DataFrames.



# Combining DataFrames with Pandas - The merge function

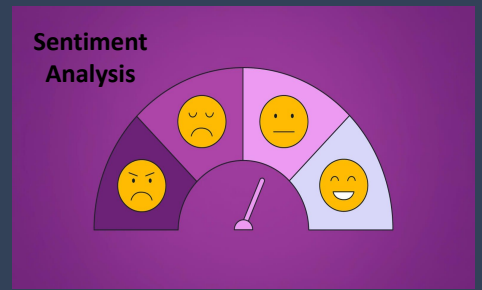
- Merging is the process of combining DataFrames based on common columns or indexes.
- It is often used to combine DataFrames with related information, similar to how SQL JOIN operation works.
- Merging allows us to specify which columns to use as keys for merging and how to handle overlapping column names.



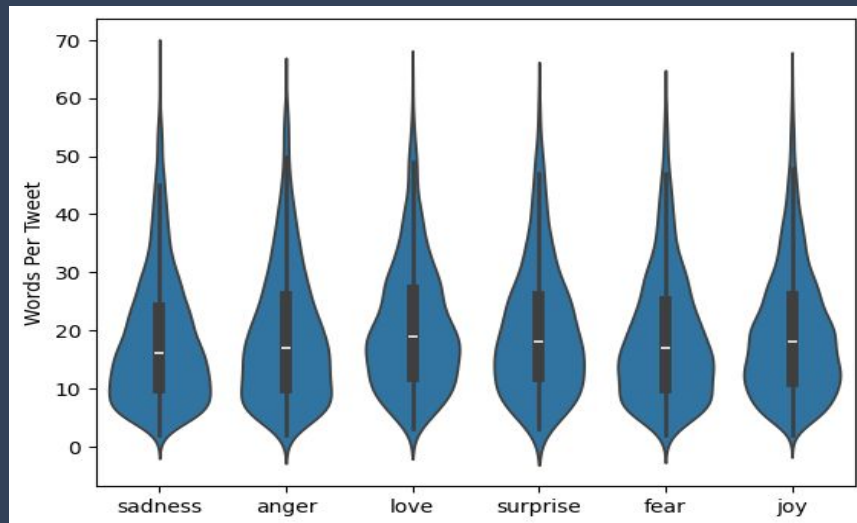
# How Does This Help Me?

## Exploratory Data Analysis ( EDA )

- Understands data patterns and relationships, critical for feature selection.
- Identifies key features, outliers, and missing values, ensuring data quality.

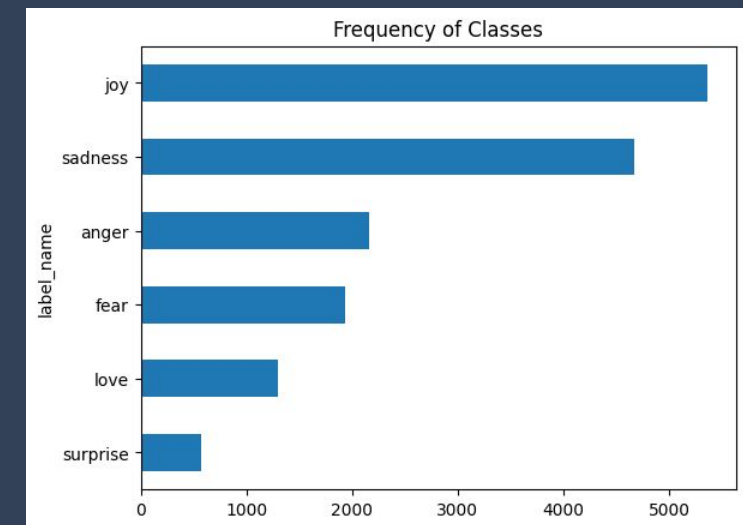


```
import seaborn as sns
df["Words Per Tweet"] = df["text"].str.split().apply(len)
sns.violinplot(data=df, x='label_name', y='Words Per Tweet')
plt.suptitle("")
plt.xlabel("")
plt.show()
```



```
import matplotlib.pyplot as plt

df["label_name"].value_counts(ascending=True).plot.barh()
plt.title("Frequency of Classes")
plt.show()
```



# Time to Code - Pandas



# Quiz

**Which function should be used to drop a row by index from a DataFrame?**

- A** `df.drop()`
- B** `df.delete()`
- C** `df.remove()`
- D** `df.drop_row()`



# Quiz

**Which function should be used to drop a row by index from a DataFrame?**

- A** `df.drop()`
- B** `df.delete()`
- C** `df.remove()`
- D** `df.drop_row()`

Break

Time!



# Session Summary

|  |  |  |  |
|--|--|--|--|
| Intro to Python,<br>Variables & Data Types           | Overview of Python's<br>syntax and structure                               | Introduction to integers,<br>floats, strings, and<br>booleans                  | Understanding how to<br>create and use<br>variables                      |
| Strings, List, Tuple,<br>Dictionary                  | Methods and operations to<br>manipulate string data                        | Creating and accessing lists<br>and tuples, understanding their<br>differences | Using key-value pairs for<br>efficient data storage and<br>retrieval     |
| If-else,<br>Loops,<br>Functions, Lambda<br>Functions | Implementing conditional<br>statements (if-else) and loops<br>(for, while) | Defining and using functions,<br>understanding parameters and<br>return values | Introduction to anonymous<br>functions for concise<br>operations         |
| Class, Modules,<br>Packages, Virtual<br>Environment  | Basics of classes and<br>objects, defining attributes<br>and methods       | Organizing code into<br>reusable modules and<br>packages                       | Setting up and managing<br>isolated Python environments<br>for projects. |

# Next Steps

- [Revise the concepts](#) learnt in today's session through class recording and post-class videos
- Attempt the [Assignment & Quiz questions](#) available on Uplevel
- Register for [Technical Coaching Session](#) through Xpert Connect option on Uplevel
- Attend the [Assignment Review Session](#) to get a walkthrough of the assignment solution

# What's Coming Next in Python Crash Course - Part 2?

Introduction to Python Libraries Essential for Generative AI

- Scikit-Learn
- Deep Learning Background
- Tensors & PyTorch

The scope of this module would be The Foundational Essentials of the most commonly used libraries in the field of Generative AI - enough to help you navigate through the Applied GenAI course!

Thank  
you