Introduction to Artificial Intelligence using Python

Week 1 of 4

Anass B. El-Yaagoubi

STAT - CEMSE, King Abdullah University of Science and Technology

KAUST Academy: AI Summer School

Thuwal, July 2nd, 2023

Course Overview

- Goal of this course is to introduce you to artificial intelligence (AI).
- Learn how to implement various AI algorithms in Python.
- Four weeks program.
  1. Python programming
  2. Data wrangling and visualization
  3. Machine learning
  4. Neural networks and deep learning
- AI project in parallel the course

Overview of week 1:

I Introduction and Python IDE setup

II Review of Python basics

III Object-Oriented Programming (OOP)

IV Working with modules and libraries

V Python Best Practices

Day 1: Introduction and Python IDE setup

- What is Artificial Intelligence?
- Why does it matter?
- What are the domains of application?

What is AI?

Fig. 1: Artificial Intelligence in the media.

AI or Data Science

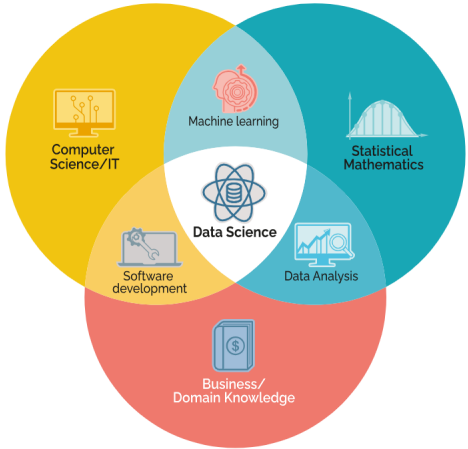

Fig. 2: Data Science is an interdisciplinary field that aims to create value from extensive collections of structured and unstructured data.

Why Does AI Matter?

- AI enables automation (saving time and effort).
- AI enhances decision-making (analyze vast amounts of data).
- AI improves processes (enhance productivity and efficiency).
- AI drives innovation (push the boundaries of what is possible).

Applications of AI: Healthcare

- Diagnosis.
- Personalized medicine.
- Drug discovery.
- Brain computer interface.

Applications of AI: Finance

- Fraud detection.
- Algorithmic trading.
- Risk assessment.

Applications of AI: Transportation

- Autonomous vehicles.
- Traffic management systems.
- Predictive maintenance.

Applications of AI: E-commerce

- Personalized recommendations.
- Customer support.
- Demand forecasting.
- Market pricing and auctions.

Applications of AI: Cinema

- Computer-generated visual effects.
- Content analysis for film production.
- Recommendation systems.

Why Python?

Fig. 3: How to learn Python?

Rich Ecosystem of Libraries for AI and ML

- TensorFlow.
- PyTorch.
- scikit-learn.
- Keras.
- Natural Language Toolkit (NLTK).
- OpenCV.

Easy Prototyping and Experimentation in AI Projects

- Interactive Development (Jupyter notebooks).
- Readability and Expressiveness (complex AI projects with multiple collaborators).
- Large Collection of Pretrained Models.

Broad Community Support

- Active Online Communities.
- Code Sharing and Reusability.
- Open Source Culture.
- Educational Resources.
- Conferences and Meetups.
- Hackathons and Competitions.

Seamless Integration with Other Technologies

- Databases.
- Cloud Platforms.
- Big Data Tools.
- Web Development.
- Visualization Tools.
- DevOps and Automation.

Scalability and Performance

- Enhanced Libraries that optimize numerical computations.
- High-Performance Computing.
- GPU Acceleration.
- Distributed Computing.
- Integration with Low-Level Languages (e.g., C and C++).

Industry Adoption of Python

- Google: Search, Maps, Translation etc.
- Facebook: Natural language processing and computer vision.
- Microsoft: Azure, Chatbots etc.
- IBM: Watson?
- Netflix: Recommendation systems.
- Amazon: Product recommendations, demand forecasting, and inventory management.
- Uber: Route optimization and driver assignment.
- Airbnb: Fraud detection, pricing optimization.

Job Opportunities in AI

- Python Proficiency: Essential skill in AI roles
- Widely Used in AI: De facto language for AI and ML
- High Demand: Job market favors Python-proficient AI professionals

Integrated Development Environment (IDE) Setup

- Install Python.
- Install Anaconda.
- Quick demonstration.
- Hands-on session.

Hands-on sessions material

Fig. 4: Scan the QR code to access the hands-on sessions material at: github.com/A-EL-YAAGOUBI/Introduction-AI/.

Day 2: Review of Python basics

- Recap of lists and their operations in Python.
- Exploring tuples and their immutability.
- Understanding dictionaries and their key-value pairs.
- Looping through lists and dictionaries.

What is a Python list?

- Ordered.
- Mutable.
- Heterogeneous.
- Indexable.
- Variable length.
- Iterable.

What can we do with lists?

```
# Example: Working with Lists
numbers = [5, 2.0, 'q', 3, 5, "Hi"]
# Length of the list
length = len(numbers)
print("Length:", length)
# Append an item to the list
numbers.append(7)
print("After append:", numbers)
# Sort the list
numbers.sort()
print("Sorted:", numbers)
# Reverse the list
numbers.reverse()
print("Reversed:", numbers)
```

What is a Python tuple?

- Immutable.
- Ordered.
- Heterogeneous.
- Indexable.
- Iterable.

What can we do with tuples?

```
# Example: Working with Tuples
my_tuple = (5, 3, 9, 1, 7)
# Length of the tuple
length = len(my_tuple)
print("Length of the tuple:", length)
# Maximum value in the tuple
maximum = max(my_tuple)
print("Maximum value in the tuple:", maximum)
# Minimum value in the tuple
minimum = min(my_tuple)
print("Minimum value in the tuple:", minimum)
# Sum of all elements in the tuple
total = sum(my_tuple)
print("Sum of all elements in the tuple:", total)
```

Slicing lists and tuples in Python:

- Slicing is done using square bracket notation: `my_list[start:end]`.
- Start index is inclusive, end index is exclusive.
- Negative indices can be used to count from the end.
- Omitting start or end indices includes all elements.
- Optional step value allows skipping elements: `my_list[start:end:step]`.
- Tuples are immutable, so slicing creates a new tuple.

Examples:

```
# Slicing a list
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
positive_slice = my_list[1:8:2]
print(positive_slice)  # Output: ?
negative_slice = my_list[8:1:-1]
print(negative_slice)  # Output: ?

# Slicing a tuple
my_tuple = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
positive_slice = my_tuple[2:9:3]
print(positive_slice)  # Output: ?
negative_slice = my_tuple[9:2:-2]
print(negative_slice)  # Output: ?
```

What is a Python dictionary?

- Key-Value Pairs.
- Keys are unique.
- Mutable.
- Unordered.
- Dynamic Size.
- Fast Lookup.

What can we do with dictionaries?

```
# Dictionary example
occurrences = {'a': 1, 'B': 3, 'z': 7}
# Get the number of key-value pairs
count = len(occurrences)
print("Number of elements:", count)
# Get all the keys in the dictionary
keys = occurrences.keys()
print("Keys:", keys)
# Get all the values in the dictionary
values = occurrences.values()
print("Values:", values)
# Clear the dictionary
occurrences.clear()
print("Cleared dictionary:", occurrences)
```

Looping through lists and dictionaries

```python
# Looping through a list
for number in numbers:
    print(number)

# Looping through a dictionary (using keys)
print("Student Scores:")
for elem in occurrences:
    print(f"Charac.: {elem}, nb of occurrences: {occurrences[elem]}")

# Looping through a dictionary (using items)
print("Student Scores:")
for characer, count in occurrences.items():
    print(f"Character: {characer}, number of occurrences: {count}")
```

Day 3: Object-Oriented Programming (OOP)

- What is procedural programming?
- What are the limitations?
- What is OOP?
- What are the advantages in AI development?

Some limitations of Procedural Programming

- Uses functions to perform actions on data
- Difficult to manage large codebases and complex relationships
- Prone to code duplication and maintenance issues
- Limited reusability and scalability

Advantages of OOP in AI Development

- Object-Oriented Programming (OOP):
  - Organizes code around objects and their interactions
  - Promotes code reusability and modularity
  - Enables building complex AI systems with manageable code
  - Facilitates collaboration and team development
- In AI development, OOP allows for:
  - Extending and modifying models through inheritance
  - Implementing complex AI architectures and algorithms
  - Facilitating code maintenance and scalability

Class vs Object in Python

- Class: Blueprint or template for creating objects.
- Object: Instance of a class with its own unique state and behavior.

Class vs Object in Python

```
class NeuralNetwork:
    input_size = 10
    hidden_size = 20
    output_size = 1

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

model = NeuralNetwork(32 x 32, 1000, 10)
```

Inheritance

- Inheritance allows a class to inherit attributes and methods from a parent class.
- Child class (subclass) inherits properties of the parent class (superclass) and can override or extend its behavior.
- Enables code reuse and promotes modularity in object-oriented programming.

Inheritance

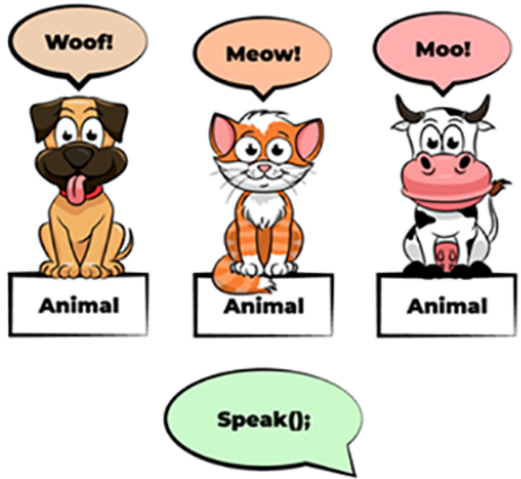

Fig. 5: Inheritance illustration.

Inheritance

```
class BaseModel:
    def train(self, data):
        # Implementation of training logic

class NeuralNetwork(BaseModel):
    def train(self, data):
        # Implementation of training logic specific to NN

model = NeuralNetwork()
model.train(training_data)
```

Polymorphism

- Polymorphism allows objects of different classes to be treated as objects of a common base class.
- Objects can be used interchangeably, and their behavior can vary based on the specific class implementation.
- Enables writing flexible and reusable code by leveraging abstract base classes and interfaces.

Polymorphism

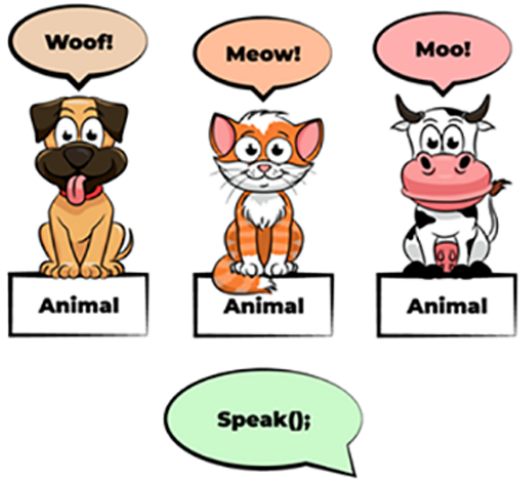

Fig. 6: Polymorphism illustration.

Polymorphism

```
class BaseModel:
    def predict(self, data):
        pass
class DecisionTree(BaseModel):
    def predict(self, data):
        # Implementation of prediction logic using Decision Tree
class NeuralNetwork(BaseModel):
    def predict(self, data):
        # Implementation of prediction logic using Neural Network

decision_tree_model = DecisionTree()
neural_network_model = NeuralNetwork()
decision_tree_prediction = decision_tree_model.predict(test_data)
neural_network_prediction = neural_network_model.predict(test_data)
```

Day 4: Working with modules and libraries in Python

- **Importing Modules**: Use `import` to bring in external modules.
- **Library Functions**: Access pre-written functions for specific tasks.
- **Namespace**: Modules create a separate namespace, preventing naming conflicts.
- **Module Installation**: Use package managers like `pip` for easy installation.
- **Documentation**: Libraries often have comprehensive documentation.

Importing Modules

```python
# Importing a module
import math

# Using a function from the math module
print(math.sqrt(16))  # Output: ?

# Importing a module with an alias
import numpy as np

# Using a function from the numpy module
arr = np.array([1, 2, 3])
print(arr)  # Output: ?
```

Library Functions

```python
# Using library functions
import random

# Generating a random integer
random_number = random.randint(1, 10)
print(random_number)  # Output: ?

# Using library functions from the math module
import math

# Calculating the factorial of a number
factorial = math.factorial(5)
print(factorial)  # Output: ?
```

Namespace

```python
# Importing specific functions from a module
from math import sqrt

# Using the imported functions directly
print(sqrt(25))  # Output: ?

# Importing all functions from a module
from numpy import *

# Using functions from the imported module without prefix
arr = array([1, 2, 3])
print(arr)  # Output: ?
```

Module Installation

```
# Installing a module using pip
# Open the terminal and run:
pip install module_name
# In Jupyter run:
!pip install module_name

# Importing the installed module
import module_name

# Using functions from the installed module
result = module_name.function_name(params)
```

Documentation

```
import math

# Viewing the documentation of the math module
print(math.__doc__)

# Accessing documentation of a function
print(math.sqrt.__doc__)

# Accessing documentation of a module using help()
help(math)

# Use the '?' sign after a module name or function
print?
pow?
```

Day 5: Python Best practices

- Use descriptive variable and function names
- Follow the PEP 8 style guide for code formatting
- Write modular and reusable code
- Handle exceptions with try-except blocks
- Document your code with clear comments and docstrings
- Optimize performance when necessary, but prioritize readability
- Continuously improve your code over time

Use Descriptive Variable and Function Names

```
# Not the best
x = get_data()
y = classify(x)

# Much better
features_x = get_data()
predictions = classify(input_data)
```

Follow the PEP 8 Style Guide

```
# Bad Example
x=5+2  # No spaces around the '+' operator
variablename=10  # No underscores between words

# Good Example
x = 5 + 2  # Spaces around the '+' operator
variable_name = 10  # Underscores between words
```

Handle Exceptions with Try-Except Blocks

```
# Bad Example
try:
    result = perform_ai_training()
except:
    print("Error occurred!")

# Good Example
try:
    result = perform_ai_training()
except Exception as e:
    print("Error occurred:", str(e))
```

Document Your Code with Comments and Docstrings

```
# Docstring example
def calculate_average(numbers):
    '''
    Calculate the average of a list of numbers.
    Args:
        numbers (list): A list of numbers.
    Returns:
        float: The average of the numbers.
    '''
    total = sum(numbers)
    average = total / len(numbers)
    return average
```