Week 1, Lecture 1

Python Fundamentals for AI

Lecture Overview

Python is the de facto language for artificial intelligence and machine learning. This lecture covers essential Python concepts needed for AI development, including data structures, control flow, functions, and object-oriented programming.

Topics Covered:

- Python basics: variables, data types, operators
- Data structures: lists, tuples, dictionaries, sets
- Control flow: if/else, loops, comprehensions
- Functions and lambda expressions
- Object-oriented programming basics
- File I/O and error handling

1. Python Basics

1.1 Variables and Data Types

```
# Basic data types in Python
print("Python Data Types")
print("="*50)

# Integers
x = 10
print(f"Integer: x = {x}, type = {type(x)}")

# Floats
y = 3.14
print(f"Float: y = {y}, type = {type(y)}")

# Strings
name = "Alice"
print(f"String: name = '{name}', type = {type(name)}")

# Booleans
is_student = True
print(f"Boolean: is_student = {is_student}, type = {type(is_student)}")

# Type conversion
num_str = "42"
num_int = int(num_str)
print(f"\nType conversion: '{num_str}' -> {num_int}")

# Multiple assignment
a, b, c = 1, 2, 3
print(f"Multiple assignment: a={a}, b={b}, c={c}")
```

```
Python Data Types
==================================================
Integer: x = 10, type = <class 'int'>
Float: y = 3.14, type = <class 'float'>
String: name = 'Alice', type = <class 'str'>
Boolean: is_student = True, type = <class 'bool'>

Type conversion: '42' -> 42
Multiple assignment: a=1, b=2, c=3
```

1.2 Operators

```
# Arithmetic operators
print("Arithmetic Operators")
print("="*50)

a, b = 10, 3
print(f"a = {a}, b = {b}")
print(f"Addition: a + b = {a + b}")
print(f"Subtraction: a - b = {a - b}")
print(f"Multiplication: a * b = {a * b}")
print(f"Division: a / b = {a / b}")
print(f"Floor division: a // b = {a // b}")
print(f"Modulo: a % b = {a % b}")
print(f"Power: a ** b = {a ** b}")

# Comparison operators
print("\nComparison Operators")
```

```
print("="*50)
print(f"a == b: {a == b}")
print(f"a != b: {a != b}")
print(f"a > b: {a > b}")
print(f"a < b: {a < b}")
print(f"a >= b: {a >= b}")
print(f"a <= b: {a <= b}")

# Logical operators
print("\nLogical Operators")
print("="*50)
x, y = True, False
print(f"x = {x}, y = {y}")
print(f"x and y: {x and y}")
print(f"x or y: {x or y}")
print(f"not x: {not x}")
```

```
Arithmetic Operators
==================================================
a = 10, b = 3
Addition: a + b = 13
Subtraction: a - b = 7
Multiplication: a * b = 30
Division: a / b = 3.3333333333333335
Floor division: a // b = 3
Modulo: a % b = 1
Power: a ** b = 1000

Comparison Operators
==================================================
a == b: False
a != b: True
a > b: True
a < b: False
a >= b: True
a <= b: False

Logical Operators
==================================================
x = True, y = False
x and y: False
x or y: True
not x: False
```

2. Data Structures

2.1 Lists

```
# Lists - ordered, mutable collections
print("Python Lists")
print("="*50)

# Creating lists
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True]
print(f"Numbers: {numbers}")
print(f"Mixed types: {mixed}")

# Indexing (zero-based)
print(f"\nIndexing:")
print(f"First element: numbers[0] = {numbers[0]}")
print(f"Last element: numbers[-1] = {numbers[-1]}")
print(f"Slice [1:4]: numbers[1:4] = {numbers[1:4]}")

# List operations
print(f"\nList Operations:")
numbers.append(6)
print(f"After append(6): {numbers}")

numbers.insert(0, 0)
print(f"After insert(0, 0): {numbers}")

numbers.remove(3)
print(f"After remove(3): {numbers}")

popped = numbers.pop()
print(f"Popped element: {popped}")
print(f"After pop(): {numbers}")

# List methods
print(f"\nList Methods:")
print(f"Length: len(numbers) = {len(numbers)}")
print(f"Sum: sum(numbers) = {sum(numbers)}")
print(f"Max: max(numbers) = {max(numbers)}")
print(f"Min: min(numbers) = {min(numbers)}")

# List comprehension
squares = [x**2 for x in range(5)]
print(f"\nList comprehension [x**2 for x in range(5)]: {squares}")

evens = [x for x in range(10) if x % 2 == 0]
print(f"Conditional comprehension: {evens}")
```

```
Python Lists
==================================================
Numbers: [1, 2, 3, 4, 5]
Mixed types: [1, 'hello', 3.14, True]

Indexing:
First element: numbers[0] = 1
Last element: numbers[-1] = 5
Slice [1:4]: numbers[1:4] = [2, 3, 4]

List Operations:
After append(6): [1, 2, 3, 4, 5, 6]
```

```
After insert(0, 0): [0, 1, 2, 3, 4, 5, 6]
After remove(3): [0, 1, 2, 4, 5, 6]
Popped element: 6
After pop(): [0, 1, 2, 4, 5]

List Methods:
Length: len(numbers) = 5
Sum: sum(numbers) = 12
Max: max(numbers) = 5
Min: min(numbers) = 0

List comprehension [x**2 for x in range(5)]: [0, 1, 4, 9, 16]
Conditional comprehension: [0, 2, 4, 6, 8]
```

2.2 Dictionaries

```
# Dictionaries - key-value pairs
print("Python Dictionaries")
print("="*50)

# Creating dictionaries
student = {
    "name": "Alice",
    "age": 20,
    "major": "Computer Science",
    "gpa": 3.8
}

print(f"Student dict: {student}")

# Accessing values
print(f"\nAccessing values:")
print(f"Name: {student['name']}")
print(f"Age: {student['age']}")
print(f"GPA: {student.get('gpa')}")
print(f"Grade (doesn't exist): {student.get('grade', 'N/A')}")

# Modifying dictionaries
print(f"\nModifying:")
student['age'] = 21
print(f"After updating age: {student['age']}")

student['courses'] = ['AI', 'ML', 'DL']
print(f"After adding courses: {student['courses']}")

# Dictionary methods
print(f"\nDictionary methods:")
print(f"Keys: {list(student.keys())}")
print(f"Values: {list(student.values())}")
print(f"Items: {list(student.items())[:2]}...")

# Dictionary comprehension
squares_dict = {x: x**2 for x in range(5)}
print(f"\nDict comprehension: {squares_dict}")

# Nested dictionaries
grades = {
    "Alice": {"math": 95, "physics": 90},
    "Bob": {"math": 88, "physics": 92}
}
print(f"\nNested dict - Alice's math: {grades['Alice']['math']}")
```

```
Python Dictionaries
==================================================
Student dict: {'name': 'Alice', 'age': 20, 'major': 'Computer Science', 'gpa': 3.8}

Accessing values:
Name: Alice
Age: 20
GPA: 3.8
Grade (doesn't exist): N/A

Modifying:
After updating age: 21
After adding courses: ['AI', 'ML', 'DL']
```

```
Dictionary methods:
Keys: ['name', 'age', 'major', 'gpa', 'courses']
Values: ['Alice', 21, 'Computer Science', 3.8, ['AI', 'ML', 'DL']]
Items: [('name', 'Alice'), ('age', 21)]...

Dict comprehension: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

Nested dict - Alice's math: 95
```

3. Control Flow

3.1 Conditional Statements

```
# If-else statements
print("Conditional Statements")
print("="*50)

score = 85

if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
else:
    grade = 'F'

print(f"Score: {score}, Grade: {grade}")

# Ternary operator
status = "Pass" if score >= 60 else "Fail"
print(f"Status: {status}")

# Multiple conditions
age = 25
has_license = True

if age >= 18 and has_license:
    print("Can drive")
elif age >= 18 and not has_license:
    print("Need license")
else:
    print("Too young")

# Checking membership
numbers = [1, 2, 3, 4, 5]
print(f"\n3 in numbers: {3 in numbers}")
print(f"10 not in numbers: {10 not in numbers}")
```

```
Conditional Statements
==================================================
Score: 85, Grade: B
Status: Pass
Can drive

3 in numbers: True
10 not in numbers: True
```

3.2 Loops

```
# For loops
print("For Loops")
print("="*50)

# Iterating over list
fruits = ["apple", "banana", "cherry"]
print("Fruits:")
for fruit in fruits:
    print(f"  - {fruit}")
```

```
# Range
print("\nRange(5):")
for i in range(5):
    print(i, end=" ")
print()

# Enumerate
print("\nEnumerate:")
for idx, fruit in enumerate(fruits):
    print(f"  {idx}: {fruit}")

# While loops
print("\nWhile Loop:")
count = 0
while count < 5:
    print(count, end=" ")
    count += 1
print()

# Loop control
print("\nLoop Control (break/continue):")
for i in range(10):
    if i == 3:
        continue  # Skip 3
    if i == 7:
        break     # Stop at 7
    print(i, end=" ")
print()

# Nested loops
print("\nNested loops (multiplication table):")
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i}x{j}={i*j}", end="  ")
    print()
```

```
For Loops
==================================================
Fruits:
  - apple
  - banana
  - cherry

Range(5):
0 1 2 3 4

Enumerate:
  0: apple
  1: banana
  2: cherry

While Loop:
0 1 2 3 4

Loop Control (break/continue):
0 1 2 4 5 6

Nested loops (multiplication table):
1x1=1  1x2=2  1x3=3
2x1=2  2x2=4  2x3=6
3x1=3  3x2=6  3x3=9
```

4. Functions

```
# Defining functions
print("Python Functions")
print("="*50)

def greet(name):
    """Simple function with one parameter"""
    return f"Hello, {name}!"

print(greet("Alice"))

# Multiple parameters and default values
def calculate_area(length, width=1):
    """Calculate rectangle area with default width"""
    return length * width

print(f"Area(5, 3): {calculate_area(5, 3)}")
print(f"Area(5): {calculate_area(5)}")  # Uses default width=1

# Multiple return values
def get_stats(numbers):
    """Return min, max, and average"""
    return min(numbers), max(numbers), sum(numbers)/len(numbers)

nums = [1, 2, 3, 4, 5]
min_val, max_val, avg = get_stats(nums)
print(f"\nStats of {nums}:")
print(f"  Min: {min_val}, Max: {max_val}, Avg: {avg}")

# *args and **kwargs
def print_args(*args, **kwargs):
    """Variable number of arguments"""
    print(f"  Args: {args}")
    print(f"  Kwargs: {kwargs}")

print("\nVariable arguments:")
print_args(1, 2, 3, name="Alice", age=20)

# Lambda functions
square = lambda x: x**2
print(f"\nLambda - square(5): {square(5)}")

# Map, filter, reduce
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(f"Map (square): {squared}")
print(f"Filter (evens): {evens}")
```

```
Python Functions
==================================================
Hello, Alice!
Area(5, 3): 15
Area(5): 5

Stats of [1, 2, 3, 4, 5]:
  Min: 1, Max: 5, Avg: 3.0

Variable arguments:
  Args: (1, 2, 3)
```

```
  Kwargs: {'name': 'Alice', 'age': 20}

Lambda - square(5): 25
Map (square): [1, 4, 9, 16, 25]
Filter (evens): [2, 4]
```

5. Object-Oriented Programming

```
# Classes and objects
print("Object-Oriented Programming")
print("="*50)

class Student:
    """Simple student class"""

    # Class variable
    school = "IUGB"

    def __init__(self, name, age):
        """Constructor"""
        self.name = name  # Instance variables
        self.age = age
        self.courses = []

    def add_course(self, course):
        """Add a course"""
        self.courses.append(course)

    def get_info(self):
        """Return student info"""
        return f"{self.name} ({self.age}) - Courses: {len(self.courses)}"

    def __str__(self):
        """String representation"""
        return f"Student: {self.name}"

# Create objects
alice = Student("Alice", 20)
bob = Student("Bob", 21)

print(f"Created: {alice}")
print(f"Created: {bob}")

# Use methods
alice.add_course("AI")
alice.add_course("ML")
bob.add_course("DL")

print(f"\n{alice.get_info()}")
print(f"{bob.get_info()}")

# Class variable
print(f"\nSchool: {Student.school}")

# Inheritance
class GraduateStudent(Student):
    """Graduate student inherits from Student"""

    def __init__(self, name, age, thesis_topic):
        super().__init__(name, age)
        self.thesis_topic = thesis_topic

    def get_info(self):
        """Override parent method"""
```

```
        base_info = super().get_info()
        return f"{base_info}, Thesis: {self.thesis_topic}"

grad = GraduateStudent("Charlie", 24, "Deep Learning")
print(f"\n{grad.get_info()}")
```

```
Object-Oriented Programming
==================================================
Created: Student: Alice
Created: Student: Bob

Alice (20) - Courses: 2
Bob (21) - Courses: 1

School: IUGB

Charlie (24) - Courses: 0, Thesis: Deep Learning
```

Summary

Key Takeaways:

- Python uses dynamic typing and clean syntax
- Lists and dictionaries are fundamental data structures
- List/dict comprehensions provide concise syntax
- Functions support default args, *args, **kwargs
- OOP with classes enables code organization
- Lambda functions for simple operations

Practice Exercises

1. Create a function that finds all prime numbers up to n

2. Implement a class for a bank account with deposit/withdraw

3. Use list comprehension to find all palindromes in a list of words

4. Write a function that flattens a nested list