Python for Machine Learning

A Comprehensive Introduction

Sayan Chaki

March 12, 2025

Outline

1. Introduction to Python
2. Python Basics
3. Control Flow
4. Functions
5. Data Structures
6. Modules and Packages

What is Python?

- High-level, interpreted programming language
- Created by Guido van Rossum in 1991
- Emphasizes code readability and simplicity
- Versatile: web development, data science, AI, automation, etc.

Python Features

- Easy to learn and use
- Extensive standard library
- Cross-platform compatibility
- Dynamically typed
- Object-oriented
- Interpreted (no compilation needed)
- Strong community support

Hello World

The simplest Python program:

```python
# This is a comment
print("Hello, World!")  # Output: Hello, World!
```

- `print()` displays output to the console
- Comments start with # and are ignored by the interpreter
- Strings can be enclosed in single or double quotes

Variables and Data Types

```python
# Integer
age = 30
print(type(age))  # Output: <class ’int’>

# Float
price = 19.99
print(type(price))  # Output: <class ’float’>

# String
name = "Python"
print(type(name))  # Output: <class ’str’>

# Boolean
is_active = True
print(type(is_active))  # Output: <class ’bool’>
```

More Data Types

```python
# List (ordered, mutable collection)
fruits = ["apple", "banana", "cherry"]
print(fruits[0])  # Output: apple

# Tuple (ordered, immutable collection)
coordinates = (10.5, 20.8)
print(coordinates[1])  # Output: 20.8

# Dictionary (key-value pairs)
person = {"name": "Alice", "age": 25}
print(person["name"])  # Output: Alice

# Set (unordered collection of unique items)
unique_numbers = {1, 2, 3, 3, 4, 4, 5}
print(unique_numbers)  # Output: {1, 2, 3, 4, 5}
```

Basic Operations

```python
# Arithmetic operations
x = 10
y = 3
print(x + y)  # Addition: 13
print(x - y)  # Subtraction: 7
print(x * y)  # Multiplication: 30
print(x / y)  # Division: 3.3333...
print(x // y)  # Floor division: 3
print(x % y)  # Modulus (remainder): 1
print(x ** y)  # Exponentiation: 1000

# String operations
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name  # Concatenation
print(full_name)  # Output: John Doe
print(first_name * 3)  # Output: JohnJohnJohn
```

Conditional Statements

```python
age = 18

# Simple if statement
if age >= 18:
    print("You are an adult")

# if-else statement
if age >= 18:
    print("You can vote")
else:
    print("You cannot vote yet")

# if-elif-else statement
if age < 13:
    print("Child")
elif age < 18:
    print("Teenager")
else:
    print("Adult")
```

Loops

For Loop:

```python
# Iterating through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Using range()
for i in range(5):  # 0, 1, 2, 3, 4
    print(i)

# Iterating through a dictionary
person = {"name": "Alice", "age": 25}
for key, value in person.items():
    print(f"{key}: {value}")
```

While Loop

```python
# Basic while loop
count = 0
while count < 5:
    print(count)
    count += 1

# Break statement
num = 0
while True:
    print(num)
    num += 1
    if num >= 5:
        break  # Exit the loop when num reaches 5

# Continue statement
for i in range(10):
    if i % 2 == 0:
        continue  # Skip even numbers
    print(i)  # Print only odd numbers
```

Defining Functions

```python
# Simple function
def greet():
    print("Hello, World!")

greet()  # Output: Hello, World!

# Function with parameters
def greet_person(name):
    print(f"Hello, {name}!")

greet_person("Alice")  # Output: Hello, Alice!

# Function with default parameter
def greet_with_title(name, title="Mr."):
    print(f"Hello, {title} {name}!")

greet_with_title("Smith")  # Output: Hello, Mr. Smith!
greet_with_title("Johnson", "Dr.")  # Output: Hello, Dr. Johnson!
```

Return Values

```python
# Function that returns a value
def add(a, b):
    return a + b

result = add(5, 3)
print(result)  # Output: 8

# Multiple return values
def get_min_max(numbers):
    return min(numbers), max(numbers)

minimum, maximum = get_min_max([5, 3, 8, 1, 10])
print(f"Min: {minimum}, Max: {maximum}")  # Output: Min: 1, Max: 10

# Early return
def is_even(num):
    if num % 2 == 0:
        return True
    return False

print(is_even(4))  # Output: True
```

Introduction to Lambda Functions

- A lambda function is an anonymous function in Python.
- It is created using the ‘lambda‘ keyword and can take multiple arguments.
- Useful for short, simple functions where defining a full function using ‘def‘ is unnecessary.

```python
# Lambda (anonymous) function
square = lambda x: x ** 2
print(square(5))  # Output: 25

# Equivalent function using def
def square_func(x):
    return x ** 2
print(square_func(5))  # Output: 25
```

Using Lambda with `map()`

- The ‘map()‘ function applies a given function to each element of an iterable (e.g., list, tuple).
- A lambda function is often used inside ‘map()‘ for simple transformations.

```python
numbers = [1, 2, 3, 4, 5]

# Using map with lambda
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16, 25]

# Equivalent using list comprehension
squared = [x ** 2 for x in numbers]
print(squared)  # Output: [1, 4, 9, 16, 25]
```

Using Lambda with `filter()`

- The ‘filter()‘ function filters elements of an iterable based on a condition.
- A lambda function is commonly used to define the filtering condition.

```python
numbers = [1, 2, 3, 4, 5]

# Filtering even numbers
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4]

# Equivalent using list comprehension
even_numbers = [x for x in numbers if x % 2 == 0]
print(even_numbers)  # Output: [2, 4]
```

Sorting with Lambda Functions

- The ‘sorted()‘ function can use a key function to determine sorting order.
- A lambda function is often used as the key to sort based on specific attributes.

```python
students = [
    {"name": "Alice", "grade": 85},
    {"name": "Bob", "grade": 92},
    {"name": "Charlie", "grade": 78}
]

# Sorting by grade
sorted_students = sorted(students, key=lambda s: s["grade"])
print(sorted_students)
```

- Equivalent using ‘operator‘ module:

```python
from operator import itemgetter
sorted_students = sorted(students, key=itemgetter("grade"))
print(sorted_students)
```

Using Lambda with `reduce()`

- The ‘reduce()‘ function (from ‘functools‘) applies a binary function cumulatively to elements in an iterable.
- This is useful for operations like summation, product, or finding the maximum.

```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]

# Finding the product of all elements
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 120

# Equivalent using a loop
product = 1
for num in numbers:
    product *= num
print(product)  # Output: 120
```

Introduction to Lists

```python
# Creating a list
fruits = ["apple", "banana", "cherry"]
```

A list in Python is an ordered collection of items. It can hold different data types like strings, numbers, and other objects. Lists are mutable, meaning you can change their content.

- Lists are created using square brackets `[]`. - Items inside the list are separated by commas.

Accessing Elements in a List

```python
# Accessing elements
print(fruits[0])  # First element: apple
print(fruits[-1])  # Last element: cherry
```

You can access individual elements of a list by indexing:
- Positive indexing starts from 0, so `fruits[0]` gives the first element. - Negative indexing starts from -1 (last element), so `fruits[-1]` gives the last element.

List Slicing

```python
# Slicing
print(fruits[1:3])  # Elements from index 1 to 2: ['banana',
    'cherry']
```

Slicing is used to get a subset of the list:
- The syntax is `list[start:end]`, where `start` is the index to begin from (inclusive) and `end` is the index to end (exclusive). - This returns a new list, and the original list is unchanged.

List Methods

```python
# Common methods
fruits.append("orange")  # Add to end
fruits.insert(1, "blueberry")  # Insert at position
fruits.remove("banana")  # Remove by value
popped = fruits.pop(1)  # Remove by index and return
print(fruits)  # Current list
print(len(fruits))  # Length of list
```

Common list operations:

- `append()` adds an item to the end. - `insert()` inserts an item at a specific position. - `remove()` removes the first occurrence of a value. - `pop()` removes an item at a specified index and returns it. - `len()` returns the number of elements in the list.

List Comprehension

```python
# List comprehension
numbers = [1, 2, 3, 4, 5]
squares = [x**2 for x in numbers if x % 2 == 0]
print(squares)  # Output: [4, 16]
```

List comprehension provides a concise way to create lists:

- The syntax is `[expression for item in iterable if condition]`. - You can apply conditions and transformations to the list elements.

Introduction to Dictionaries

```python
# Creating a dictionary
person = {
    "name": "Alice",
    "age": 25,
    "city": "New York"
}
```

A dictionary is an unordered collection of key-value pairs. It is also mutable. In a dictionary:

- Keys must be unique and are typically strings or numbers. - Values can be of any data type, including lists and other dictionaries. - Dictionaries are created using curly braces `{}`.

Accessing Values in a Dictionary

```python
# Accessing values
print(person["name"])  # Output: Alice
print(person.get("phone", "Not found"))  # Safe access with default
```

You can access the values of a dictionary using the keys:
- `person["name"]` retrieves the value for the key `"name"`. - The `get()` method is safer because it returns a default value if the key is not found.

Adding and Updating Items in a Dictionary

```python
# Adding/updating key-value pairs
person["email"] = "alice@example.com"  # Add new key
person["age"] = 26  # Update existing key
```

You can add new key-value pairs or update existing ones:

- person["email"] = "alice@example.com" adds a new key "email". - person["age"] = 26 updates the value for the existing key "age".

Removing Items from a Dictionary

```python
# Removing items
del person["city"]  # Remove by key
popped = person.pop("email")  # Remove and return value
```

You can remove items from a dictionary:

- `del person["city"]` deletes the key-value pair for `"city"`. - `pop()` removes a key-value pair and returns the value.

Dictionary Methods

```python
# Useful methods
print(person.keys())  # Get all keys
print(person.values())  # Get all values
print(person.items())  # Get all key-value pairs
```

Some useful methods in dictionaries:

- `keys()` returns all the keys. - `values()` returns all the values. - `items()` returns all the key-value pairs.

Dictionary Comprehension

```python
# Dictionary comprehension
squares = {x: x**2 for x in range(1, 6)}
print(squares)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Dictionary comprehension is similar to list comprehension:

- The syntax is `{key: value for item in iterable if condition}`. - It allows you to create dictionaries in a compact form.

Functions in Python

```python
# Defining a function
def greet(name):
    print(f"Hello, {name}!")

# Calling the function
greet("Alice")  # Output: Hello, Alice!
```

A function in Python is a block of code designed to perform a specific task. Functions are defined using the `def` keyword and can take parameters. Once defined, the function can be called, passing the necessary arguments. Functions can:

- Accept parameters (e.g., `name` in the `greet` function). - Execute a block of code. - Return a result using the `return` keyword.

Returning Values from Functions

```python
# Function with return value
def add(a, b):
    return a + b

result = add(3, 5)  # result = 8
print(result)
```

Functions can return values. The `return` keyword is used to send a result back to the caller. This can be assigned to a variable (e.g., `result` in the above code).

- `add(3, 5)` returns the sum of 3 and 5. - The result is stored in `result` and printed.

Introduction to Classes in Python

```python
# Defining a class
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound.")

# Creating an instance of the class
animal = Animal("Dog")
animal.speak()  # Output: Dog makes a sound.
```

A class in Python is a blueprint for creating objects (instances). A class encapsulates data and methods that operate on that data.
- A class is defined using the `class` keyword. - The `__init__()` method is the constructor method that initializes the object's attributes. - Methods like `speak()` define behaviors that can be performed on the object.

Creating Objects from a Class

```python
# Creating objects from a class
dog = Animal("Dog")
cat = Animal("Cat")

dog.speak()  # Output: Dog makes a sound.
cat.speak()  # Output: Cat makes a sound.
```

Once a class is defined, you can create multiple objects from it:
- Each object is an instance of the class and can have its own state (e.g., `dog.name` is ”Dog”, `cat.name` is ”Cat”). - Objects can call methods defined in the class (e.g., `dog.speak()` and `cat.speak()`).

Inheritance in Python

```python
# Defining a subclass
class Dog(Animal):
    def speak(self):
        print(f"{self.name} barks.")

# Creating an instance of the subclass
dog = Dog("Rex")
dog.speak()  # Output: Rex barks.
```

Inheritance allows one class to inherit the attributes and methods of another class. In the example above:
- The `Dog` class inherits from the `Animal` class. - This means the `Dog` class can access the `__init__()` method and other methods (like `speak()`) from the `Animal` class. - The `Dog` class can also override inherited methods to provide specific behavior.
In this case, `Dog` overrides the `speak()` method to provide a more specific behavior (”barks”).

Encapsulation in Python

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age  # Private attribute

    def get_age(self):
        return self.__age

person = Person("Alice", 30)
print(person.get_age())  # Output: 30
```

Encapsulation refers to the concept of restricting access to some of an object's attributes and methods. In Python:

- Attributes with a double underscore prefix (e.g., `__age`) are considered private. - These private attributes cannot be accessed directly from outside the class. - Instead, public methods like `get_age()` provide controlled access to private data.

This helps protect data from being accidentally modified and allows the class to control how the data is accessed.

Importing Modules

```python
# Importing entire module
import math
print(math.sqrt(16))  # Output: 4.0

# Importing specific functions
from math import sqrt, pi
print(sqrt(16))  # Output: 4.0
print(pi)  # Output: 3.141592653589793

# Importing with alias
import math as m
print(m.cos(0))  # Output: 1.0

# Importing all functions (not recommended)
from random import *
print(randint(1, 10))  # Random number between 1 and 10
```

Standard library modules include:

- `math`: mathematical functions
- `random`: random number generation
- `datetime`: date and time handling

Creating Your Own Module

File: mymodule.py

```python
# Define variables
PI = 3.14159

# Define functions
def square(x):
    return x ** 2

def cube(x):
    return x ** 3

# Define a class
class Calculator:
    def add(self, a, b):
        return a + b

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