

Introduction to Python

Tan Sein Jone

University of British Columbia

August 7, 2024

Table of contents

1. Object Oriented Programming
2. Pythonics
3. Data Handling

Classes and Objects

- A class is a blueprint for creating object
- Classes define the properties and behaviours of objects
- Objects have attributes and methods
- Attributes are variables that store data
- Methods are functions that perform actions

Examples of Classes in Pandas

- `pd.DataFrame` is a class that represents a two-dimensional table of data
- `pd.Series` is a class that represents a one-dimensional array of data
- `pd.Index` is a class that represents an index of data

Examples of Methods in Pandas

- `pd.DataFrame.head()` returns the first n rows of a dataframe
- `pd.DataFrame.tail()` returns the last n rows of a dataframe
- `pd.DataFrame.describe()` returns the summary statistics of a dataframe

Attributes

- Attributes are variables that store data
- They are defined in the `__init__` method
- They are accessed using the dot operator

Breakdown

- `class Person:` defines a class named `Person`
- `def __init__(self, name, age):` defines a method named `__init__` that initializes the object
- `self` is a reference to the object itself
- `self.name` and `self.age` are attributes of the object

Attributes

```
person = Person("John", 36)
print(person.name)
print(person.age)
```

Methods

- Methods are functions that perform actions
- They are defined in the class
- They are accessed using the dot operator

Methods

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello , my_name_is {self.name}")
```

Breakdown

- `class Person:` defines a class named `Person`
- `def __init__(self, name, age):` defines a method named `__init__` that initializes the object
- `self` is a reference to the object itself
- `self.name` and `self.age` are attributes of the object
- `def greet(self):` defines a method named `greet` that prints a greeting

Encapsulation

- Encapsulation is the bundling of data and methods that operate on the data
- It restricts access to some of the object's components
- It prevents the accidental modification of data

Inheritance

- Inheritance is the mechanism of basing a class upon another class
- It allows a class to inherit attributes and methods from another class
- It allows a class to override methods of another class

Polymorphism

- Polymorphism is the ability to present the same interface for different data types
- It allows a function to accept different data types
- It allows a class to override methods of another class

Do I Need to Know How to Write Classes?

- For the purposes of the program, probably not
- Why did I cover this topic? Because it is important to understand how Python works under the hood
- When you call a method in Pandas, you are calling a method of a class
- When you create a dataframe in Pandas, you are creating an object of a class
- This understanding will make it much easier to debug and troubleshoot your code

Summary

- Classes are blueprints for creating objects
- Objects have attributes and methods
- Attributes are variables that store data
- Methods are functions that perform actions
- Encapsulation is the bundling of data and methods
- Inheritance is the mechanism of basing a class upon another class
- Polymorphism is the ability to present the same interface for different data types

Pythonics

- Pythonic code is code that follows the conventions of the Python language
- It is code that is clean, readable, and maintainable
- It is code that is idiomatic and expressive

Zen of Python

- The Zen of Python is a collection of aphorisms that capture the philosophy of Python
- It is a set of guiding principles for writing computer programs
- It is a set of rules for writing Pythonic code

Zen of Python

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- There should be one— and preferably only one —obvious way to do it
- Now is better than never
- Although never is often better than right now

Value Swapping and Multiple Assignment

- Python allows you to swap the values of two variables in a single line
- It also allows you to assign multiple values to multiple variables in a single line

Breakdown

- `a, b = b, a` swaps the values of `a` and `b`
- `print(a, b)` prints the values of `a` and `b`
- The output is 2 1

List Slicing

- Python allows you to slice lists using the slice operator
- It allows you to slice lists using the start, stop, and step arguments
- It allows you to slice lists using negative indices

List Slicing

```
numbers = [1, 2, 3, 4, 5]
print(numbers[1:3])
print(numbers[:2])
print(numbers[::-1])
```

Breakdown

- `print(numbers[1:3])` slices the list from index 1 to index 3
- `print(numbers[:2])` slices the list with a step of 2
- `print(numbers[::-1])` slices the list in reverse order
- The output is `[2, 3]`, `[1, 3, 5]`, `[5, 4, 3, 2, 1]`

Passing Multiple Arguments

- Python allows you to pass multiple arguments to a function
- It also allows you to pass keyword arguments to a function

Passing Multiple Arguments

```
def greet(*names):  
    for name in names:  
        print(f"Hello , {name}")  
  
greet("John" , "Jane" , "Jack")
```

Breakdown

- `def greet(*names):` defines a function named `greet` that takes multiple arguments
- `for name in names:` iterates over the arguments
- `print(f"Hello, name")` prints a greeting for each argument
- The output is `Hello, John, Hello, Jane, Hello, Jack`

List Comprehension

- List comprehension is a concise way to create lists
- It allows you to create lists using a single line of code
- It is more readable and expressive than traditional loops

List Comprehension

```
squares = [x ** 2 for x in range(10)]  
print(squares)
```

Breakdown

- `squares = [x ** 2 for x in range(10)]` creates a list of squares
- `print(squares)` prints the list of squares
- The output is `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

List Comprehension vs Loops

- It is possible to achieve the same result using a loop
- However, list comprehension is more concise and expressive
- It is also more readable and maintainable
- It is the preferred way to create lists in Python
- If the list comprehension is too complex, use a loop instead

Lambda Functions

- Lambda functions are anonymous functions
- They are defined using the `lambda` keyword
- They are used to create small, one-line functions

Lambda Functions

```
add = lambda x, y: x + y  
print(add(1, 2))
```


Breakdown

- `add = lambda x, y: x + y` defines a lambda function that adds two numbers
- `print(add(1, 2))` calls the lambda function with arguments 1 and 2
- The output is 3

A Note on Indentation

- Python uses indentation to define blocks of code
- It uses whitespace to delimit code

A Note on Indentation

- Indentation is important in Python
- It is used to define the scope of code
- It is used to group statements together

Summary

- Pythonic code is code that follows the conventions of the Python language
- It is code that is clean, readable, and maintainable
- It is code that is idiomatic and expressive
- The Zen of Python is a collection of aphorisms that capture the philosophy of Python
- It is a set of guiding principles for writing computer programs
- It is a set of rules for writing Pythonic code

1. Object Oriented Programming
2. Pythonics
3. Data Handling

Data Handling

- Data handling is the process of managing data
- It involves reading, writing, and processing data
- It involves working with files, databases, and APIs

Lists and Dictionaries

- Lists are ordered collections of items
- Dictionaries are unordered collections of key-value pairs
- Lists are indexed by integers
- Dictionaries are indexed by keys

Combining Lists and Dictionaries

- You can combine lists and dictionaries to create complex data structures
- You can nest lists and dictionaries to create hierarchical data structures

Combining Lists and Dictionaries

```
person = {  
    "name": "John",  
    "age": 36,  
    "friends": ["Jane", "Jack"]  
}  
print(person["name"])  
print(person["age"])  
print(person["friends"])
```

Breakdown

- `person = {"name": "John", "age": 36, "friends": ["Jane", "Jack"]}` creates a dictionary
- `print(person["name"])` prints the value of the key "name"
- `print(person["age"])` prints the value of the key "age"
- `print(person["friends"])` prints the value of the key "friends"
- The output is John, 36, ["Jane", "Jack"]

Reading and Writing Files

- Python allows you to read and write files
- It allows you to open files in read mode, write mode, or append mode
- It allows you to read files line by line or all at once

Reading and Writing Files

```
with open("data.txt", "w") as file:  
    file.write("Hello ,_world!")
```

```
with open("data.txt", "r") as file:  
    data = file.read()  
    print(data)
```

Reading and Writing Files

- Python allows you to read and write files
- It allows you to open files in read mode, write mode, or append mode
- It allows you to read files line by line or all at once

NumPy

- NumPy is a library for numerical computing
- It provides support for arrays and matrices
- It allows you to perform mathematical operations on arrays and matrices
- It is the foundation of many other libraries

NumPy

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = a * b
print(c)

d = np.dot(a, b)
print(d)
```

Breakdown

- `a = np.array([1, 2, 3])` creates an array `a`
- `b = np.array([4, 5, 6])` creates an array `b`
- `c = a * b` multiplies the arrays `a` and `b` element-wise
- `print(c)` prints the result of the multiplication
- `d = np.dot(a, b)` computes the dot product of the arrays `a` and `b`
- `print(d)` prints the result of the dot product

Pandas

- Pandas is a library for data manipulation and analysis
- It provides support for data structures like Series and DataFrame
- It allows you to read and write data from various sources
- It is built on top of NumPy

Pandas IO

- Pandas allows you to read and write data from various sources
- It allows you to read and write data from CSV files, Excel files, SQL databases, and APIs
- It allows you to read and write data from URLs, HTML tables, and clipboard

Pandas IO

```
import pandas as pd

data = pd.read_csv("data.csv")
print(data)

data.to_csv("data.csv", index=False)
```

Breakdown

- `data = pd.read_csv("data.csv")` reads a CSV file into a dataframe
- `print(data)` prints the dataframe
- `data.to_csv("data.csv", index=False)` writes the dataframe to a CSV file

A Note on File Pathing in Linux

- In Linux, the Python Interpreter knows to look in the current directory for files
- If you are running Python in a different directory, you can use a relative path to the file
- If you are running Python in a different directory, you can use an absolute path to the file

Manipulating Dataframes

- Pandas allows you to manipulate dataframes
- It allows you to filter, sort, group, and aggregate data
- It allows you to merge, join, and concatenate data
- It allows you to reshape, pivot, and melt data

Manipulating Dataframes

```
import pandas as pd

data = pd.read_csv("age.csv")
data = data[data["age"] > 30]
data = data.sort_values("age")
data['rank'] = data['age'].rank()
data.loc['total'] = data.sum()
print(data)
```

Breakdown

- `data = pd.read_csv("age.csv")` reads a CSV file into a dataframe
- `data = data[data["age"] > 30]` filters the dataframe by age
- `data = data.sort_values("age")` sorts the dataframe by age
- `data['rank'] = data['age'].rank()` ranks the dataframe by age
- `data.loc['total'] = data.sum()` sums the dataframe
- `print(data)` prints the dataframe

pd.apply()

- Pandas allows you to apply functions to dataframes
- It allows you to apply functions to rows, columns, or cells
- It allows you to apply lambda functions, user-defined functions, or built-in functions

pd.apply()

```
import pandas as pd

data = pd.read_csv("age.csv")
data['age'] = data['age'].apply(lambda x: x + 1)
print(data)
```

Breakdown

- `data = pd.read_csv("age.csv")` reads a CSV file into a dataframe
- `data['age'] = data['age'].apply(lambda x: x + 1)` applies a lambda function to the age column
- `print(data)` prints the dataframe

Merge and Join

- Pandas allows you to merge and join dataframes
- It allows you to merge dataframes on columns or indices
- It allows you to merge dataframes using inner, outer, left, or right joins

Merge and Join

```
import pandas as pd

data1 = pd.read_csv("age.csv")
data2 = pd.read_csv("blood_type.csv")
data = pd.merge(data1, data2, on="name")
print(data)
```

Breakdown

- `data1 = pd.read_csv("age.csv")` reads a CSV file into a dataframe
- `data2 = pd.read_csv("blood_type.csv")` reads a CSV file into a dataframe
- `data = pd.merge(data1, data2, on="name")` merges the dataframes on the "name" column
- `print(data)` prints the merged dataframe

Null Values

- Pandas allows you to handle null values
- It allows you to drop null values, fill null values, or interpolate null values
- It allows you to check for null values, count null values, or filter null values

Null Values

```
import pandas as pd

data = pd.read_csv("temp.csv")
data = data.dropna()
data = data.fillna(0)
data = data.interpolate()
print(data)
```


Breakdown

- `data = pd.read_csv("temp.csv")` reads a CSV file into a dataframe
- `data = data.dropna()` drops null values from the dataframe
- `data = data.fillna(0)` fills null values with 0
- `data = data.interpolate()` interpolates null values
- `print(data)` prints the dataframe

Rules of Thumb for Missing Data

- If the missing data is random, drop the rows
- If the missing data is systematic, fill the missing values
- If the missing data is time-dependent, interpolate the missing values

Best Practices for Missing Data

- Regardless of what you do with missing data, always document your decisions
- Always check for missing data before performing any analysis
- Always check for missing data after performing any analysis

Summary

- Data handling is the process of managing data
- It involves reading, writing, and processing data
- It involves working with files, databases, and APIs
- Lists are ordered collections of items
- Dictionaries are unordered collections of key-value pairs
- NumPy is a library for numerical computing
- Pandas is a library for data manipulation and analysis