05 Python Modules

01 py_compile

- **Automatic Compilation** : Python compiles scripts to bytecode before running them, creating `.pyc` files
- **Manual Compilation** : Use `py_compile` and `compileall` modules

```
import py_compile
```

```
py_compile.compile('abc.py')
```

```
python -m compileall .
```

`__init__.py` File

- Required to make Python treat directories as packages
- Can be empty or execute initialization code for the package

---

02 RE

- Used to filter text or text strings in programming languages.
- Useful for data cleaning, wrangling, and performing various text-processing tasks in data science & machine learning

Purposes

- **Parsing** : Identifying & extracting text matching certain patterns
- **Searching** : Locating substrings with specific criteria
- **Replacing** : Substituting matching substrings with other strings
- **Splitting** : Dividing strings based on matching patterns
- **Validation** : Checking if text meets specified criteria

Identifiers

- \d = any number
- \D = anything but a number
- \s = space
- \S = anything but a space
- \w = any letter
- \W = anything but a letter
- . = any character, except for a new line
- \b = space around whole words
- . = period. must use backslash, because . normally means any character

```
import re
```

```
text = "start learning"
pattern = re.compile("start") # Compiles a regex pattern into a regex
object
```

```
result = pattern.match(text)  # Checks for a match only at the beginning
of the string
```

```
line = "He is a German called Mayer"
if re.search(r"M[ae][iy]er", line): # Searches the entire string for a
match
        print("I found one!")
```

```
text = "Mona and Reena are two sisters, of age 12 and 15 respectively"
ages = re.findall(r'\d{1,3}', text)
print(ages)                          # ['12', '15']
```

```
re.split(r'\s*', 'here are some words')
# Output: ['here', 'are', 'some', 'words']
```

```
text = "Cats are smarter than dogs"
re.sub(r'dogs', 'cats', text)      # Output: "Cats are smarter than cats"
```

```
matches = re.finditer(r"([a-zA-Z]+) \d+", "June 24, August 9, Dec 12")
for match in matches:
        print("Match at index: %s, %s" % (match.start(), match.end()))
```

```
# Match at index: 0, 7
# Match at index: 9, 17
# Match at index: 19, 25
```

```
match.group()
match.start()
match.end()
```

---

03 PDB

- `pdb` stands for Python Debugger
- It is a built-in module in Python used for interactive debugging of Python programs
- It allows you to set breakpoints, step through code, inspect variables, and evaluate expressions at runtime

```
import pdb
```

```
(Pdb) help
(Pdb) help clear
pdb.help()      # Prints general help
```

```
(Pdb) where
(Pdb) bt        # Prints a stack trace, with the most recent frame at the
bottom
```

```
(Pdb) down      # Move the current frame one level down in the stack trace
```

```
(Pdb) up        # Move the current frame one level up in the stack trace
```

```
(Pdb) break 10  # Sets a breakpoint at the specified line number or
function
(Pdb) break mymodule.py:10
```

```
(Pdb) clear     # Clears breakpoints
(Pdb) clear 1
```

```
(Pdb) disable 1
```

```
(Pdb) enable 1
```

```
(Pdb) step      # Execute the current line, stop at the first possible occasion
```

```
(Pdb) next      # Continue execution until the next line in the current function is reached
```

```
(Pdb) jump 20   # Set the next line that will be executed
```

```
(Pdb) list 5,10 # List source code for the current file
```

```
(Pdb) display a # Use to display the value of a when it changes
```

---

04 OS

- The OS module in python provides functions for interacting with the OS
- Comes under Python's standard utility modules
- Provides a portable way of using OS dependent functionality

```
import os
```

```
os.name # Name of the operating system-dependent module
```

```
print(os.getcwd())  # Current working directory path
```

```
print(os.listdir('.')) # List of files and directories in the current directory
```

```
os.chdir('..') # Changes to the parent directory
```

```
os.mkdir('test_directory') # A new directory named 'test_directory' is created
```

```
os.remove('test_file.txt') # The file 'test_file.txt' is removed
```

```
os.rename('old.txt','new.txt') # 'old.txt' is renamed to 'new.txt'
```

---

05 Sys

- Provides functions and variables used to manipulate different parts of the Python runtime environment

```
import sys
```

```
sys.ps1  # '>>> '
sys.ps2  # '... '
```

```
print(sys.argv[0], sys.argv[1], sys.argv[2])
$ python script.py arg1 arg2
# Output: script.py arg1 arg2
```

```
sys.maxsize # 9223372036854775807
```

```
sys.stdout = open('output.txt', 'w')
sys.stdin, sys.stderr
```

```
sys.getsizeof(obj) # size of an object in bytes
```

```
sys.path # path to the current-directory
```

```
sys.platform # 'linux'
```

```
sys.version
```

```
# '3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]'
```

```
sys.platform # 'linux'
```

```
sys.exit(0)
```

---

06 NumPy (Numerical Python)

- Library that supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays

ndarray

- The most important object defined in NumPy is an N-dimensional array type
- It describes the collection of items of the same type
- Each element in ndarray is an object of data-type object (called dtype)

```
import numpy as np
```

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A)

[[1 2 3]
 [4 5 6]]
```

```
A = np.arange(10)
# [0 1 2 3 4 5 6 7 8 9]
```

```
B = np.linspace(0, 1, 5)
# [0.   0.25 0.5  0.75 1.  ]
```

```
C = np.random.randint(100, size=5)
# [25 62 24 81 39]  # This will vary each time
```

```
D = np.random.choice([3,5,6,7,9,2], size=(3,5))
[[3 2 5 2 6]
[5 9 3 6 9]
[5 6 9 3 3]]
```

```
E = np.random.shuffle([1,2,3,4,5])
# [4 1 3 5 2]
```

```
A = np.array([1, 2, 3])
B = A + 10
print(B)
# [11 12 13]
```

```
B = A * 2
print(B)
# [2 4 6]
```

```
A = np.arange(10)
print(E[2:5])
# [2 3 4]
```

Matrix Operations

- Addition `A + B`
- Subtraction `A - C`
- Multiplication (Element-wise matrix multiplication or the Hadamard product) `A * B`
- Transpose `A.T or A.transpose()`
- Inverse `inv(A)`
- Trace `trace(A)`
- Determinant `det(A)`

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)
print(C)

[[19 22]
 [43 50]]
```

ndarray Object – Parameters

ndarray.ndim

- The number of dimensions (axes) of the array

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A.ndim)
# 2
```

ndarray.shape

- For a matrix with `n` rows and `m` columns, the shape will be `(n, m)`

```
B = np.array([[1, 2, 3], [4, 5, 6]])
print(B.shape)
# (2, 3)
```

ndarray.size

- The total number of elements in the array

```
C = np.array([[1, 2, 3], [4, 5, 6]])
print(C.size)
# 6
```

ndarray.dtype

- The data type of the elements in the array

```
D = np.array([1, 2, 3])
print(D.dtype)
# int64 - This may vary depending on the platform
```

ndarray.itemsize

- The size (in bytes) of each element in the array

```
E = np.array([1, 2, 3], dtype=np.int32)
print(E.itemsize)
# 4
```

ndarray.nbytes

- The total number of bytes consumed by the elements of the array

```
F = np.array([1, 2, 3], dtype=np.int32)
print(F.nbytes)
# 12
```

ndarray.T

- The transposed array (swapping rows and columns)
- Equivalent to `ndarray.transpose()`

```
G = np.array([[1, 2, 3], [4, 5, 6]])
print(G.T)

[[1 4]
 [2 5]
 [3 6]]
```

Broadcasting

- Ability of NumPy to treat arrays of different shapes during arithmetic operations
- If the dimensions of two arrays are dissimilar, operations on arrays of non-similar shapes is still possible in NumPy
- The smaller array is broadcast to the size of the larger array so that they have compatible shapes

---

07 matplotlib

- Plotting library for creating static, animated, and interactive visualizations in Python
- Grids and Legends
- Bar plot `plt.bar(x,y)`
- Histogram `plt.hist(var)`
- Scatter plot `plt.scatter(x,y`
- Stem plot `plt.stem(x, y,use_line_collection=True)`
- Pie Plot `pie(data)`

- Subplots with in the same plot `subplot(2,2,1)`
- Ticks in Plot : Ticks are the values used to show specific points on the coordinate axis `plt.xticks(np.arange(start, end, tick))`

```
import matplotlib.pyplot as plt
```

```
plt.plot(x, y, label='line')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.show()
```

---

08 Pandas (Panal Data and Python Data Analysis)

- Python package that offers various data structures and operations for manipulating numerical data and time series
- Two data structure for manipulating data
  - Series
  - DataFrame
- **DataFrame Object** : 2D, size-mutable, potentially heterogeneous tabular data
- **Series Object** : 1D, labeled array capable of holding any data type
- **Data Alignment** : Automatic data alignment with labeled data
- **Missing Data Handling** : Gracefully handles missing data
- **Reshaping** : Flexible reshaping and pivoting of datasets
- **Group By** : Split-apply-combine methodology for data aggregation and transformation
- **Time Series** : Powerful tools for working with time-series data

```
import pandas as pd
```

Series

- Array-like object containing an array of data and an associated array of data labels, called its index
- Pandas Series is nothing but a column in an excel sheet

```
# Creating a Series
data = [1, 3, 5, 7, 9]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)
print(series)
# Output
a    1
b    3
c    5
d    7
e    9
dtype: int64
```

Attributes & Methods of Series

```
# Creating a Series
data = [10, 20, 30, 40, 50]
series = pd.Series(data)
```

```
series[0]   # First element: 10
series[2:4] # 20 30
```

```
series.head() # default 5
0 10
1 20
2 30
3 40
4 50
dtype: int64
```

```
series.tail()
0 10
1 20
2 30
3 40
4 50
dtype: int64
```

```
series.axes # [RangeIndex(start=0, stop=5, step=1)]
```

```
series.dtype # `int64`
```

```
series.empty # False
```

```
series.ndim # 1
```

```
series.size # 5
```

```
series.values # [10 20 30 40 50]
```

```
series.index  # RangeIndex(start=0, stop=5, step=1)
```

```
# Arithmetic Operations
series + 10
0 20
1 30
2 40
3 50
4 60
series * 2
0 20
1 40
2 60
3 80
4 100
```

```
np.sqrt(series)
0 3.162278
1 4.472136
2 5.477226
3 6.324555
4 7.071068
```

```
# Conditional Selection
series[series > 30]
3 40
4 50
```

```
# Series Alignment
series1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
series2 = pd.Series([4, 5, 6], index=['b', 'c', 'd'])
series1 + series2

a NaN
```

```
b 6.0
c 8.0
d NaN
dtype: float64
```

```
# Handling Missing Data
data = {'a': 1, 'b': 2, 'c': None}
series = pd.Series(data)

series.isna()
a False
b False
c True
dtype: bool

series.fillna(0)
a 1.0
b 2.0
c 0.0
dtype: float64
```

DataFrame

- A `DataFrame` is a two-dimensional labeled data structure with columns of potentially different types

Creating a DataFrame

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 27, 22, 32],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston
```

Dealing with Rows & Columns

- add, delete & rename rows and columns in a DataFrame

```
df['Country'] = 'USA'
print(df)
```

```
      Name  Age         City Country
0    Alice   24     New York     USA
1      Bob   27  Los Angeles     USA
2  Charlie   22      Chicago     USA
3    David   32      Houston     USA
```

```
df = df.drop('Country', axis=1)
print(df)
```

```
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston
```

```
df = df.rename(columns={'Name':'Full Name', 'City':'Location'})
print(df)
```

```
   Full Name  Age     Location
0      Alice   24     New York
1        Bob   27  Los Angeles
2    Charlie   22      Chicago
3      David   32      Houston
```

```
new_row = {'Full Name': 'Eve', 'Age': 29, 'Location': 'Miami'}
df = df.append(new_row, ignore_index=True)
print(df)
```

```
   Full Name  Age     Location
0      Alice   24     New York
1        Bob   27  Los Angeles
2    Charlie   22      Chicago
3      David   32      Houston
4        Eve   29        Miami
```

```
df = df.drop(0)
print(df)
```

```
   Full Name  Age     Location
1        Bob   27  Los Angeles
2    Charlie   22      Chicago
3      David   32      Houston
4        Eve   29        Miami
```

Indexing & Selecting Data

- Select data by label or by position using `.loc` and `.iloc`.

```
df.loc[1:3, ['Full Name', 'Age']]
# df.iloc[2:4 ,['Full Name', 'Age']]
```

```
   Full Name  Age
1        Bob   27
2    Charlie   22
3      David   32
```

Working with Missing Data

```
df.at[1, 'Age'] = None
print(df.isna())
```

```
   Full Name    Age  Location
1      False   True     False
2      False  False     False
3      False  False     False
4      False  False     False
```

```
df['Age'] = df['Age'].fillna(df['Age'].mean())
print(df)
```

```
   Full Name        Age     Location
1        Bob  27.666667  Los Angeles
2    Charlie  22.000000      Chicago
```

```
3      David  32.000000      Houston
4        Eve  29.000000        Miami
```

```
df.at[1, 'Age'] = None
df = df.dropna()
print(df)
```

```
   Full Name   Age Location
2    Charlie  22.0  Chicago
3      David  32.0  Houston
4        Eve  29.0    Miami
```

Iterating over Rows and Columns

```
for index, row in df.iterrows():
    print(index, row['Full Name'], row['Age'])
```

```
2 Charlie 22.0
3 David 32.0
4 Eve 29.0
```

```
for column in df:
    print(column, df[column].values)
```

```
Full Name ['Charlie' 'David' 'Eve']
Age [22. 32. 29.]
Location ['Chicago' 'Houston' 'Miami']
```

Working with CSV Files

- **Pandas** is a powerful data manipulation and analysis library for Python
- **Manipulating Data** : Performing operations like filtering, grouping, and aggregating
- **Processing Data** : Applying functions to process data

```
import pandas as pd
df.to_csv('studdata.csv')
data = pd.read_csv('file.csv')
```