Python Examples & Info By Gudimilla Naveen.

Tuesday, May 28, 2024

12:52 PM

1. Introduction to Python

Overview of Python

- **What is Python?** A high-level, interpreted programming language known for its readability and versatility.
- **History and Versions**: Created by Guido van Rossum in 1991, with Python 2.x and 3.x being the major versions. Python 3.x is the current standard.
- **Installing Python**:
    - **Standard Python Distribution**: Python.org Download
    - **Anaconda Distribution**: Anaconda.com

- **Setting up a Development Environment**:
    - IDEs: PyCharm, VS Code, Jupyter Notebook
    - Text Editors: Sublime Text, Atom

2. Basic Concepts

Syntax and Semantics

- **Writing and Executing Scripts**: Create a file with a .py extension and run it using python filename.py.
- **Indentation**: Python uses indentation to define blocks of code.

- **Variables and Data Types**:
    - Integers (int): x = 5
    - Floats (float): y = 3.14
    - Strings (str): name = "Alice"
    - Booleans (bool): is active = True

Basic Operations

- **Arithmetic Operations**: +, -, *, /, // (floor division), % (modulus), ** (exponentiation)
- **String Operations**:
  - Concatenation: first_name + last_name
  - Repetition: "Hello" * 3
  - Slicing: name[0:3]
- **Comments**: Use # for single-line comments and """...""" for multi-line comments.

3. Control Structures

Conditional Statements

- **if, elif, else**:

```
python
Copy code
if condition1:
   # do something
elif condition2:
   # do something else
else:
   # do another thing
```

Loops

- **for Loop**:

```
python
Copy code
for i in range(5):
   print(i)
```

- **while Loop**:

```
python
Copy code
while condition:
   # do something
```

- **break and continue**:

```
python
Copy code
for i in range(10):
   if i == 5:
      break
   if i % 2 == 0:
        continue
    print(i)
```

4. Functions

Defining Functions

- **Function Syntax**:

```
python
Copy code
def function_name(parameters):
    # function body
    return value
```

- **Parameters and Return Values**:

```
python
Copy code
def add(a, b):
    return a + b
```

Built-in Functions

- **Common Built-in Functions**: print(), len(), type(), input()

Lambda Functions

- **Anonymous Functions**:

```
python
Copy code
add = lambda x, y: x + y
print(add(2, 3))
```

5. Data Structures

Lists

- **Creating Lists**: my_list = [1, 2, 3, 4, 5]
- **List Operations**:
  - append(): my_list.append(6)
  - remove(): my_list.remove(3)
  - Slicing: my_list[1:4]

Tuples

- **Creating Tuples**: my_tuple = (1, 2, 3)
- **Tuple Operations**: Accessing elements, slicing

Dictionaries

- **Creating Dictionaries**: my_dict = {'key1': 'value1', 'key2': 'value2'}
- **Dictionary Operations**:
  - Adding: my_dict['key3'] = 'value3'
  - Removing: del my_dict['key2']
  - Keys and Values: my_dict.keys(), my_dict.values()

Sets

- **Creating Sets**: my_set = {1, 2, 3, 4, 5}
- **Set Operations**:
  - Adding: my_set.add(6)
  - Removing: my_set.remove(2)
  - Union and Intersection: set1.union(set2), set1.intersection(set2)

6. Modules and Packages

Importing Modules

- **Standard Library Modules**: import math, import datetime, import os, import sys
- **Using Functions from Modules**:

  ```
  python
  Copy code
  import math
  print(math.sqrt(16))
  ```

Creating Modules

- **Writing and Importing Your Own Modules**: Create a Python file and import it using import filename.

Packages

- **Using pip to Install Packages**:

  ```
  sh
  Copy code
  pip install numpy
  ```

- **Commonly Used Packages**: NumPy, Pandas, Matplotlib

7. File Handling

Reading and Writing Files

- **Opening Files**:

  ```
  python
  Copy code
  file = open('filename.txt', 'r')
  ```

- **Reading Files**:

  ```
  python
  Copy code
  content = file.read()
  file.close()
  ```

- **Writing to Files**:

  ```
  python
  Copy code
  file = open('filename.txt', 'w')
  file.write('Hello, World!')
  file.close()
  ```

8. Exception Handling

Error Handling

- **try, except, finally Blocks**:

  ```
  python
  Copy code
  try:
      # code that may raise an exception
  except ExceptionType:
      # code to handle the exception
  finally:
      # code that will run no matter what
  ```

- **Raising Exceptions**:

  ```
  python
  Copy code
  raise ValueError("An error occurred")
  ```

9. Object-Oriented Programming (OOP)

Classes and Objects

- **Defining Classes**:

  ```
  python
  Copy code
  class MyClass:
      def __init__(self, value):
          self.value = value
  ```

- **Creating Objects**:

  ```
  python
  ```

Copy code
```
obj = MyClass(10)
```

Methods and Constructors

- **Instance Methods, Static Methods, and Class Methods**:

  python
  Copy code
  ```
  class MyClass:
      def instance_method(self):
          return 'instance method called', self

      @classmethod
      def class_method(cls):
          return 'class method called', cls

      @staticmethod
      def static_method():
          return 'static method called'
  ```

Inheritance and Polymorphism

- **Inheriting Classes**:

  python
  Copy code
  ```
  class BaseClass:
      pass

  class DerivedClass(BaseClass):
      pass
  ```

- **Overriding Methods**:

  python
  Copy code
  ```
  class BaseClass:
      def greet(self):
          print("Hello from BaseClass")

  class DerivedClass(BaseClass):
      def greet(self):
          print("Hello from DerivedClass")
  ```

10. Advanced Topics

Decorators

- **Function Decorators**:

  python

```
Copy code
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")
```

Generators and Iterators

- **Creating Iterators**:

```
python
Copy code
class MyIterator:
    def __init__(self, start, end):
        self.current = start
        self.end = end

def __iter__(self):
        return self

def __next__(self):
        if self.current >= self.end:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

- **Using Generators (yield)**:

```
python
Copy code
def my_generator():
    yield 1
    yield 2
    yield 3
```

Context Managers

- **Using with Statement**:

```
python
Copy code
with open('file.txt', 'r') as file:
    content = file.read()
```

Regular Expressions

- **Using re Module**:

```
python
Copy code
import re
pattern = re.compile(r'\d+')
result = pattern.findall('123 abc 456 def')
print(result)
```

11. Working with Data

Data Analysis

- **Introduction to NumPy and Pandas**:

```
python
Copy code
import numpy as np
import pandas as pd
```

- **Data Manipulation and Analysis**:

```
python
Copy code
data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)
```

Data Visualization

- **Introduction to Matplotlib and Seaborn**:

```
python
Copy code
import matplotlib.pyplot as plt
import seaborn as sns
```

- **Creating Plots and Visualizations**:

```
python
Copy code
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```

12. Web Development

Web Frameworks

- **Introduction to Flask and Django**:
    - Flask: Micro web framework
    - Django: Full-stack web framework

- **Building Basic Web Applications**:

```
python
Copy code
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Flask!"

if __name__ == "__main__":
    app.run(debug=True)
```

13. Testing and Debugging

Unit Testing

- **Using unittest Module**:

```
python
Copy code
import unittest

class TestMath(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(add(2, 3), 5)
```

Debugging Techniques

- **Using pdb for Debugging**:

```
python
Copy code
import pdb; pdb.set_trace()
```

14. Best Practices

Code Style

- **PEP 8 Guidelines**: PEP 8 -- Style Guide for Python Code

Version Control

- **Introduction to Git and GitHub**:
  - Git: Version control system
  - GitHub: Platform for hosting and collaborating on Git repositories

15. Project Development

Building Projects

- **Working on Small Projects to Apply Learned Concepts**:

    - To-do list
    - Simple web scraper
    - Basic web app

Each section can be studied and practiced individually. Begin with the basics and gradually move to more complex topics. Here are some resources and exercises for each section to help you learn and practice:

Resources

- **Official Python Documentation**: docs.python.org
- **Python for Beginners**: Python.org Beginners Guide
- **Interactive Learning Platforms**: Codecademy, Coursera, edX, LeetCode, HackerRank

Exercises

- **Basic Concepts**: Write simple scripts that perform arithmetic operations, manipulate strings, and use conditional statements and loops.
- **Functions**: Write functions that perform various tasks and use built-in functions.
- **Data Structures**: Practice creating and manipulating lists, tuples, dictionaries, and sets.
- **File Handling**: Write scripts to read from and write to files.
- **OOP**: Create classes, objects, and practice inheritance and polymorphism.
- **Advanced Topics**: Implement decorators, generators, and context managers.
- **Data Analysis and Visualization**: Use NumPy, Pandas, Matplotlib, and Seaborn to analyze and visualize data.
- **Web Development**: Build simple web applications using Flask or Django.
- **Testing and Debugging**: Write unit tests and use debugging techniques to troubleshoot your code.
- **Project Development**: Work on small projects to integrate and apply the concepts you've learned.