PYTHON — COMPLETE COURSE

This course starts at the very beginning and progresses through intermediate and advanced topics. Each module has examples and exercises linked from this page.

Course Outline

1. Introduction & Setup
2. Basics: Syntax, Variables, Types
3. Control Flow: Conditionals & Loops
4. Data Structures: Lists, Tuples, Sets, Dicts
5. Functions, Scope & Lambdas
6. Modules & Packages
7. Object-Oriented Programming
8. File I/O & Exceptions
9. Standard Library & pip
10. Testing, Debugging & Logging
11. Advanced Topics
12. Projects & Guided Exercises

Resources

- Examples: `Informatik/Python/examples/`
- Exercises: `Informatik/Python/exercises/`

Quick Start Example

```python
def greet(name: str) -> str:
    return f"Hello, {name}!"

if __name__ == "__main__":
    print(greet("world"))
```

Introduction to Python

Python is a versatile, high-level programming language known for its simplicity and readability. It is widely used in web development, data analysis, artificial intelligence, scientific computing, and more.

Core Concepts

1. Syntax and Variables

Python uses indentation to define blocks of code. Variables are dynamically typed, meaning you don't need to declare their type explicitly.

```python
# Example: Variables and Printing
name = "Alice"
age = 25
print(f"Name: {name}, Age: {age}")
```

2. Control Flow

Python supports conditional statements and loops to control the flow of execution.

```python
# Example: If-Else and Loops
number = 10
if number > 0:
    print("Positive number")
else:
    print("Non-positive number")

for i in range(5):
    print(i)
```

3. Data Structures

Python provides built-in data structures like lists, tuples, sets, and dictionaries to store and manipulate data.

```python
# Example: Lists and Dictionaries
fruits = ["apple", "banana", "cherry"]
prices = {"apple": 1.2, "banana": 0.5, "cherry": 2.5}

print(fruits)
print(prices["apple"])
```

4. Functions

Functions in Python are defined using the def keyword. They help in organizing code into reusable blocks.

```python
# Example: Functions
def add(a, b):
    return a + b

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

5. Object-Oriented Programming (OOP)

Python supports OOP principles like classes and inheritance.

```python
# Example: Classes and Objects
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
```

```python
        return f"Hi, I'm {self.name} and I'm {self.age} years old

person = Person("Alice", 25)
print(person.greet())
```

6. File Handling

Python makes it easy to work with files using built-in functions.

```python
# Example: Reading and Writing Files
with open("example.txt", "w") as file:
    file.write("Hello, World!")

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

7. Error Handling

Python uses try and except blocks to handle exceptions gracefully.

```python
# Example: Exception Handling
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

8. Modules and Libraries

Python has a rich ecosystem of modules and libraries. You can import built-in modules or install third-party libraries using pip.

```python
# Example: Using Modules
import math

print(math.sqrt(16))
```

9. Advanced Topics

Python also supports advanced features like decorators, generators, and context managers.

```python
# Example: Generators
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

for num in fibonacci(5):
    print(num)
```

10. Decorators

Decorators are a powerful feature in Python that allow you to modify the behavior of a function or class. They are often used for logging, enforcing access control, instrumentation, and more.

```python
# Example: Using a Decorator
def decorator(func):
    def wrapper():
        print("Before the function call")
        func()
        print("After the function call")
    return wrapper

@decorator
def say_hello():
    print("Hello!")

say_hello()
```

11. Context Managers

Context managers are used to manage resources, such as files or network connections, ensuring they are properly cleaned up after use. The with statement is commonly used for this purpose.

```python
# Example: Using a Context Manager
with open("example.txt", "w") as file:
    file.write("Hello, World!")

# The file is automatically closed after the block
```

12. Generators

Generators are a special type of iterator in Python that allow you to yield values one at a time, making them memory-efficient for large datasets.

```python
# Example: Generator Function
def countdown(n):
    while n > 0:
        yield n
        n -= 1

for number in countdown(5):
    print(number)
```

13. Regular Expressions

Regular expressions (regex) are a powerful tool for pattern matching and text processing. Python provides the re module for working with regex.

```python
# Example: Using Regular Expressions
import re

pattern = r"\b[A-Za-z]+\b"
text = "Python is amazing!"

matches = re.findall(pattern, text)
print(matches)
```

14. Working with APIs

Python makes it easy to interact with APIs using libraries like requests. You can send HTTP requests and handle responses efficiently.

```python
# Example: Fetching Data from an API
import requests
```

```python
response = requests.get("https://api.github.com")
if response.status_code == 200:
    print(response.json())
```

15. Working with Databases

Python provides libraries like sqlite3 for working with databases. You can execute SQL queries to interact with the database.

```python
# Example: SQLite Database
import sqlite3

connection = sqlite3.connect("example.db")
cursor = connection.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER, nam
cursor.execute("INSERT INTO users (id, name) VALUES (?, ?)", (1,

connection.commit()
connection.close()
```

16. Concurrency and Parallelism

Python supports concurrency and parallelism using modules like threading and multiprocessing. These are useful for running tasks simultaneously.

```python
# Example: Using Threads
import threading

def print_numbers():
    for i in range(5):
```

```
        print(i)

thread = threading.Thread(target=print_numbers)
thread.start()
thread.join()
```

17. Data Visualization

Python has powerful libraries like matplotlib and seaborn for creating visualizations.

```python
# Example: Plotting with Matplotlib
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.plot(x, y)
plt.title("Line Graph")
plt.show()
```