Advanced Python Notes

1. Object-Oriented Programming (OOP)

Python supports Object-Oriented Programming (OOP), which allows encapsulating data and functions into objects.
Concepts include Class, Object, Inheritance, Polymorphism, and Encapsulation.

Example:
class Car:
def __init__(self, brand, model):
self.brand = brand
self.model = model

def start(self):
print(f"{self.brand} {self.model} started")

my_car = Car("Tesla", "Model S")
my_car.start()

2. Exception Handling

Errors can occur during runtime. Python provides try-except blocks to handle exceptions gracefully.

Example:
try:
x = 10 / 0
except ZeroDivisionError:
print("Cannot divide by zero!")
finally:
print("Execution completed")

Custom exceptions can be created using:
class MyError(Exception):
pass

3. File Handling

Python provides easy methods to read and write files.

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

Modes include: 'r' (read), 'w' (write), 'a' (append), 'rb'/'wb' (binary).

4. Modules and Packages

Modules are Python files (.py) containing functions and variables.
Packages are directories containing multiple modules with an __init__.py file.

Example:
import math
print(math.sqrt(25))

from datetime import date
print(date.today())

5. Iterators and Generators

Iterators allow sequential access to elements. Generators simplify iterator creation using the 'yield' keyword.

Example:
```
def count_up_to(n):
i = 1
while i <= n:
yield i
i += 1
for num in count_up_to(5):
print(num)
```

Generators use less memory as they produce items one by one.

6. Decorators

Decorators modify function behavior without changing the code inside.

Example:
```
def decorator(func):
def wrapper():
print("Before function")
func()
print("After function")
return wrapper

@decorator
def hello():
print("Hello, world!")

hello()
```

7. NumPy and Pandas

NumPy is used for numerical operations, and Pandas for data analysis.

Example:
```
import numpy as np
import pandas as pd

arr = np.array([1, 2, 3])
print(arr.mean())

data = {'Name': ['A', 'B'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df)
```

8. Regular Expressions (RegEx)

Regular expressions are used for pattern matching.

Example:
```
import re
pattern = r"[A-Za-z]+"
text = "Python123Programming"
matches = re.findall(pattern, text)
print(matches)
```

Functions include: re.search(), re.match(), re.findall(), re.sub().

9. Multithreading and Multiprocessing

Multithreading allows concurrent execution of tasks.

Example:
```
import threading

def task():
print("Running in thread")

t1 = threading.Thread(target=task)
t1.start()
```

Multiprocessing uses multiple cores for parallelism.

Example:
```
from multiprocessing import Process

def show():
print("Process running")

p = Process(target=show)
p.start()
p.join()
```

10. Virtual Environments

Virtual environments isolate project dependencies.

Commands:
```
python -m venv env
source env/bin/activate (Linux/macOS)
env\Scripts\activate (Windows)
pip install package_name
```