Python Basics - Theory and Examples

1. Introduction to Python Basics

Python syntax defines the structure of the code, while semantics defines its meaning.

Python uses indentation instead of braces {}.

```
# Example: Case Sensitivity
name = "Krish"
Name = "Naik"

print(name)  # Output: Krish
print(Name)  # Output: Naik

# Example: Indentation
age = 32
if age > 30:
    print(age)

print(age)
```

2. Variables in Python

A variable is a container for storing data values.

Python automatically assigns a type based on the value assigned.

```
# Example: Declaring Variables
age = 32
height = 6.1
name = "Krish"
is_student = True

print("Age:", age)
print("Height:", height)
print("Name:", name)
```

3. Data Types in Python

Python has built-in data types like integers, floats, strings, and booleans.

It also has complex structures like lists, tuples, sets, and dictionaries.

```
# Example: Checking Data Types
age = 35
print(type(age))  # <class 'int'>

height = 5.11
print(type(height))  # <class 'float'>

name = "Krish"
print(type(name))  # <class 'str'>
```

4. Operators in Python

Operators perform operations on values.

Arithmetic operators include +, -, *, /, //, %, and **.

```
# Example: Arithmetic Operations
a = 10
b = 5

print(a + b)  # Addition -> 15
print(a - b)  # Subtraction -> 5
print(a * b)  # Multiplication -> 50
print(a / b)  # Division -> 2.0
print(a // b) # Floor Division -> 2
print(a % b)  # Modulus -> 0
print(a ** b) # Exponentiation -> 100000
```

Python Control Flow - Theory and Examples

1. Conditional Statements

Conditional statements allow the execution of different code blocks based on conditions.

Python supports `if`, `if-else`, and `elif` statements.

```
# Example: Checking Voting Eligibility
age = 20

if age >= 18:
    print("You are allowed to vote in the elections")

# Example: Checking if a Person is a Minor
age = 16

if age >= 18:
    print("You are eligible for voting")
else:
    print("You are a minor")

# Example: Categorizing Age Groups
age = 17

if age < 13:
    print("You are a child")
elif age < 18:
    print("You are a teenager")
else:
    print("You are an adult")
```

2. Loops in Python

Loops allow repeating actions multiple times. Python has `for` and `while` loops.

```
# Example: Looping Through a Range
for i in range(5):
    print(i)
```

```
# Example: Looping with Start and Step
for i in range(1, 10, 2):
    print(i)

# Example: Countdown Using `while` Loop
count = 5
while count > 0:
    print(count)
    count -= 1
```

3. Loop Control Statements

Loop control statements modify the flow of loops.

- `break` stops the loop.
- `continue` skips an iteration.
- `pass` is a placeholder.

```
# Example: Using `break` to Stop a Loop
for i in range(10):
    if i == 5:
        break
    print(i)

# Example: Using `continue` to Skip an Iteration
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Python Data Structures - Theory and Examples

1. Lists

Lists are ordered, mutable collections that can store multiple data types.

They allow indexing, slicing, and various built-in methods for manipulation.

```
# Example: Creating a List
names = ["Krish", "Jack", "Jacob", 1, 2, 3, 4, 5]
print(names)

mixed_list = [1, "Hello", 3.14, True]
print(mixed_list)

# Example: List Methods
numbers = [1, 2, 3, 4, 5]

numbers.append(6)  # Adds an element at the end
numbers.remove(3)  # Removes the first occurrence of 3
numbers.reverse()  # Reverses the list

print(numbers)
```

2. Tuples

Tuples are immutable (cannot be modified after creation).

They are ordered collections like lists but more efficient.

```
# Example: Creating a Tuple
numbers = (1, 2, 3, 4, 5)
print(numbers)

# Example: Tuple Packing and Unpacking
a, b, c = (10, 20, 30)
print(a, b, c)
```

3. Sets

Sets store unique, unordered elements.

They do not allow duplicates and support set operations like union, intersection, and difference.

```
# Example: Creating a Set
my_set = {1, 2, 3, 4, 5}
print(my_set)

# Example: Set Operations
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

print(A | B)  # Union
print(A & B)  # Intersection
print(A - B)  # Difference
```

4. Dictionaries

Dictionaries store key-value pairs.

Keys must be unique and immutable, while values can be of any type.

```
# Example: Creating a Dictionary
student = {"name": "Krish", "age": 32, "grade": "A"}
print(student)

# Example: Accessing and Modifying a Dictionary
print(student["name"])  # Access value by key

student["age"] = 33  # Modify value
student["subject"] = "Math"  # Add new key-value pair

print(student)
```

Python Functions - Theory and Examples

1. Functions

A function is a block of reusable code that performs a specific task.

Functions improve readability, organization, and reusability.

```
# Example: Function to Check Even or Odd
def even_or_odd(num):
    if num % 2 == 0:
        print("The number is even")
    else:
        print("The number is odd")

even_or_odd(24)
```

2. Lambda Functions

Lambda functions are small, anonymous functions using the `lambda` keyword.

They can have multiple arguments but only a single expression.

```
# Example: Lambda for Addition
addition = lambda a, b: a + b
print(addition(5, 6))
```

3. Map Function

The map() function applies a given function to all items in an iterable.

```
# Example: Squaring Numbers Using `map()`
def square(x):
    return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(square, numbers))

print(squared_numbers)

# Example: Using Lambda with `map()`
numbers = [1, 2, 3, 4, 5]
```

```
squared_numbers = list(map(lambda x: x * x, numbers))


print(squared_numbers)
```

4. Filter Function

The filter() function filters elements based on a condition.

```
# Example: Filtering Even Numbers
def is_even(num):
    return num % 2 == 0


numbers = [1, 2, 3, 4, 5, 6, 7, 8]
even_numbers = list(filter(is_even, numbers))


print(even_numbers)


# Example: Using Lambda with `filter()`
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
greater_than_five = list(filter(lambda x: x > 5, numbers))


print(greater_than_five)
```

Python Modules and Standard Library - Theory and Examples

1. Importing Modules

Modules are Python files containing functions, variables, and classes.

They help in organizing and reusing code efficiently.

```
# Example: Importing a Module
import math
print(math.sqrt(16))

# Example: Importing Specific Functions
from math import sqrt, pi
print(sqrt(25))
print(pi)

# Example: Using External Libraries (e.g., NumPy)
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr)
```

2. Python Standard Library

Python's Standard Library provides built-in modules for various functionalities.

Commonly used modules include `math`, `random`, and `os`.

```
# Example: Using the `math` Module
import math
print(math.sqrt(16))  # Square root
print(math.pi)        # Pi constant

# Example: Generating Random Numbers (`random` Module)
import random
print(random.randint(1, 10))  # Random integer between 1 and 10
print(random.choice(['apple', 'banana', 'cherry']))  # Random choice

# Example: Working with Directories (`os` Module)
import os
print(os.getcwd())  # Get current working directory
```

Python File Handling & Exception Handling - Theory and Examples

1. File Handling

File handling allows reading and writing files in Python.

```
# Example: Reading a File
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)

# Example: Reading a File Line by Line
with open('example.txt', 'r') as file:
    for line in file:
        print(line.strip())

# Example: Writing to a File (Overwriting)
with open('example.txt', 'w') as file:
    file.write('Hello World!\n')
    file.write('This is a new line.')

# Example: Appending to a File (Without Overwriting)
with open('example.txt', 'a') as file:
    file.write("Appending new data!\n")
```

2. File Paths

Python's `os` module helps in handling file paths.

```
# Example: Getting the Current Working Directory
import os
cwd = os.getcwd()
print(f"Current working directory is {cwd}")

# Example: Creating a New Directory
new_directory = "package"
os.mkdir(new_directory)
print(f"Directory '{new_directory}' created")
```

```
# Example: Listing Files and Directories
items = os.listdir('.')
print(items)

# Example: Joining Paths
dir_name = "folder"
file_name = "file.txt"
full_path = os.path.join(dir_name, file_name)
print(full_path)
```

3. Exception Handling

Exception handling ensures that errors are managed gracefully.

```
# Example: Basic Try-Except Block
try:
    a = b
except:
    print("The variable has not been assigned")

# Example: Handling Specific Exceptions
try:
    a = b
except NameError as ex:
    print(ex)
```

Python OOP & Advanced Concepts - Theory and Examples

1. Classes and Objects

Classes are blueprints for creating objects. Objects are instances of a class with attributes and methods.

```
# Example: Creating a Class and Object
class Car:  # Define a class named 'Car'
    pass

audi = Car()  # Create an object of the Car class
bmw = Car()   # Create another object

print(type(audi))  # Output: <class '__main__.Car'>

# Example: Adding Attributes to an Object
audi.windows = 4  # Assign an attribute dynamically
print(audi.windows)  # Output: 4
```

Note: Objects in Python can have attributes dynamically assigned to them.

This makes Python classes very flexible, but be careful with unstructured data.

2. Inheritance

Inheritance allows one class to inherit attributes and methods from another.

```
# Example: Single Inheritance
class Car:  # Parent class
    def __init__(self, windows, doors, enginetype):
        self.windows = windows  # Number of windows in the car
        self.doors = doors  # Number of doors in the car
        self.enginetype = enginetype  # Type of engine

    def drive(self):
        print(f"The person will drive the {self.enginetype} car.")

car1 = Car(4, 5, "petrol")  # Creating an object of Car class
car1.drive()  # Calling the drive method
```

```
# Example: Inheritance with a Child Class
class Tesla(Car):  # Child class inheriting from Car class
    def __init__(self, windows, doors, enginetype, is_selfdriving):
         super().__init__(windows, doors, enginetype)  # Calling the constructor
of the parent class
        self.is_selfdriving = is_selfdriving  # New attribute for Tesla class

    def selfdriving(self):
        print(f"Tesla supports self-driving: {self.is_selfdriving}")

tesla1 = Tesla(4, 5, "electric", True)  # Creating an object of Tesla class
tesla1.selfdriving()  # Calling the selfdriving method
```

Note: Inheritance allows us to reuse code from the parent class in the child class.

Use `super()` to call methods of the parent class inside the child class.

3. Iterators

Iterators allow sequential access to elements without exposing their structure.

```
# Example: Using an Iterator
my_list = [1, 2, 3, 4, 5]  # A normal list
iterator = iter(my_list)  # Convert the list into an iterator

print(next(iterator))  # Output: 1
print(next(iterator))  # Output: 2
```

Note: Iterators save memory when working with large data sets.

You can convert an iterable (like a list) into an iterator using `iter()`.

4. Generators

Generators generate values lazily using `yield`, reducing memory usage.

```
# Example: Generator Function for Squares
def square(n):  # Generator function
    for i in range(n):  # Loop from 0 to n-1
        yield i**2  # Yield square of each number

```

```
for i in square(3):  # Iterate through generator output
    print(i)  # Output: 0, 1, 4
```

Note: Generators are memory-efficient because they generate items one at a time. They are useful when dealing with large sequences of data.